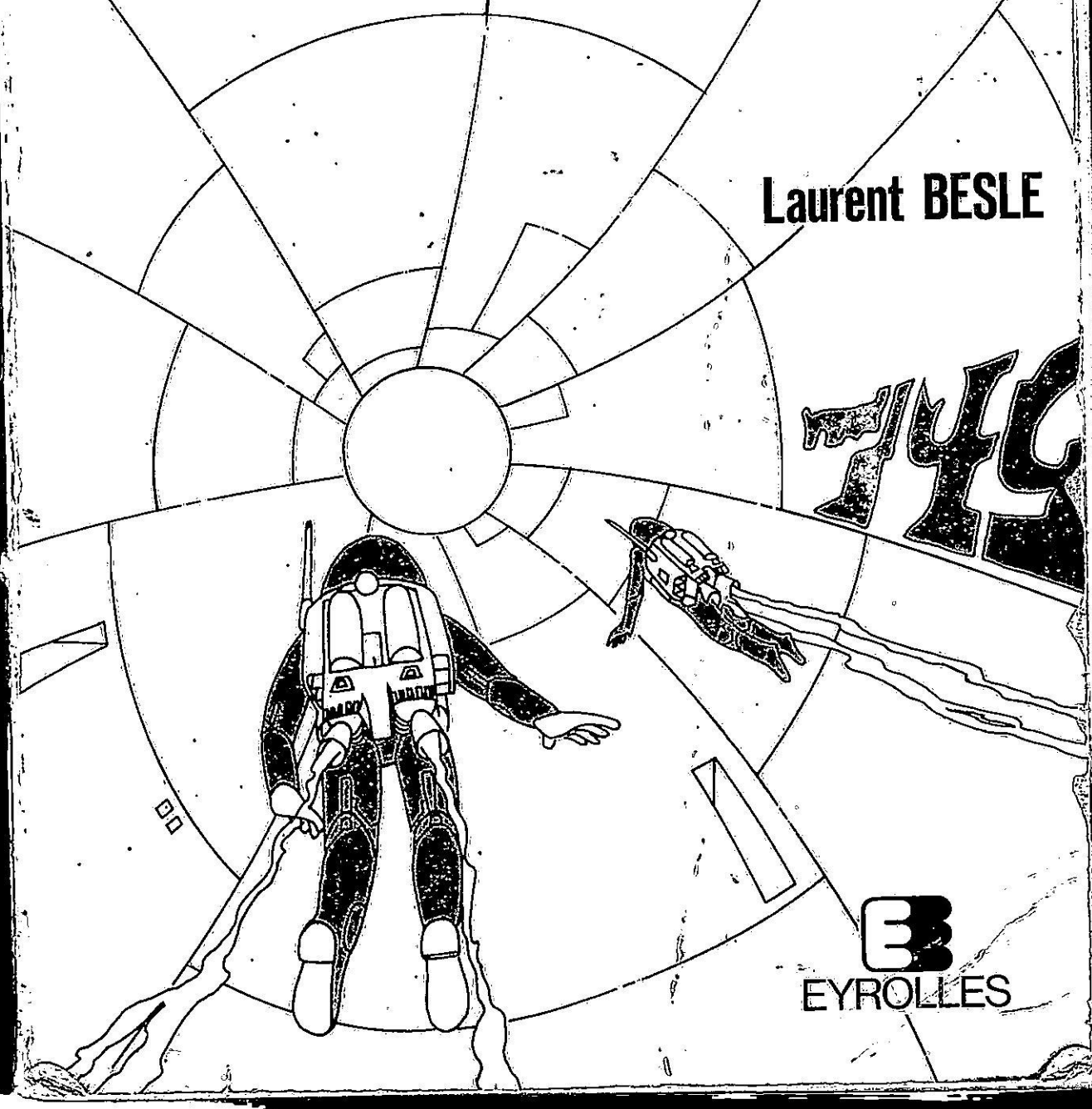
AU COEUR DE L'ATARI ST

Laurent BESLE



EYROLLES

Au cœur de l'ATARI ST

CHEZ LE MEME EDITEUR

P. Jaulent et L. Baticle	CIRCUITS PERIPHERIQUES DE LA FAMILLE 68000
P. Jaulent	LE MICROPROCESSEUR 68000 ET SA PROGRAMMATION
L. Léon	PROGRAMMATION EN ASSEMBLEUR 68000
L. Cervoni et M. Couprie	GEM : GUIDE POUR L'UTILISATION ET LA PROGRAMMATION
B. Costales	C de A à Z
Ph. Dax	LANGAGE C
K. Jensen et N. Wirth	PASCAL, MANUEL DE L'UTILISATEUR ET RAPPORT DE DEFINITION
P. Philippot	TURBO-PASCAL Procédures et fonctions pour IBM-PC et compatibles
P. Saint-Dizier	INITIATION A LA PROGRAMMATION EN PROLOG
K.L. Clark et F.G. McCabe	MICRO-PROLOG : PROGRAMMER EN LOGIQUE
A. Mevel et Th. Gueguen	SMALLTALK-80
J. Krutch	EXPERIENCES D'INTELLIGENCE ARTIFICIELLE EN BASIC

Au cœur de l'ATARI ST

par

Laurent BESLE

DEUXIEME EDITION
mise à jour et corrigée

EYROLLES

61, boulevard Saint-Germain - 75005 PARIS
1987

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Editions EYROLLES,
61, Boulevard Saint-Germain
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40). »

« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. ».

© Editions EYROLLES, 1987

INTRODUCTION

La micro-informatique constitue la « nouvelle frontière » des années 80. Si l'on exclut les connotations californiennes de l'expression, demeurent la notion de dépassement, d'aventure, de bouleversement constant des principes un temps considérés comme acquis. Pourtant, alors que tant de passionnés passent des nuits entières devant leur écran et se soumettent à une formation permanente et accélérée aux innovations constantes de la technique, beaucoup d'auteurs conservent les mêmes stéréotypes vieillis. Combien de préfaces pour vanter les caractéristiques « inouïes et sans pareilles » de la machine étudiée (alors que le lecteur en est souvent convaincu !) ou, plus froidement et plus commercialement, pour affirmer que tout lecteur, néophyte ou développeur, trouvera dans l'ouvrage ce qu'il n'a jamais trouvé ailleurs.

512 Ko de mémoire vive, 192 Ko de mémoire morte, 95 touches dont 10 de fonctions, haute résolution 640*400 points, interfaces parallèle, série, MIDI, lecteur 3,5 pouces double densité, microprocesseur 16/32 bits... naturellement les composantes techniques d'un système micro-informatique ont une importance et cet ouvrage ne faillira pas à la tradition, dans la mesure où seront étudiées chacune de ces caractéristiques. Mais le lecteur recherchant un dépliant publicitaire pourra s'adresser à son revendeur habituel, essayer un 520 ST et les machines concurrentes et décider en connaissance de cause.

Cet ouvrage s'adresse exclusivement au programmeur, de préférence en langage C ou en assembleur. A celui-ci, l'auteur à l'espoir insensé d'apporter des méthodes de programmation, à travers de multiples exemples, ainsi qu'une somme d'informations de référence. Il semblerait même que tout programmeur en C ou en assembleur 68000, sur 520 ST ou sur une autre machine, pourrait tirer quelque profit des pages qui suivent, mais peut-être y a-t-il là excès de forfanterie ou même tartufferie !

Un panorama des instructions de base du langage C et de l'assembleur 68000 est fourni dans cet ouvrage, celui-ci ne saurait cependant suffire au programmeur débutant dans l'un de ces langages, auquel il est conseillé de se reporter aux ouvrages de référence en la matière, dont la bibliographie en annexe fournit un aperçu.

Ce livre a été écrit sans précipitation excessive, afin de tenir compte des évolutions habituelles lors du lancement d'une machine. Certains compilateurs C et assembleurs annoncés n'étant malheureusement pas encore disponibles lors de la rédaction de ces pages, les exemples fournis ont été principalement développés sur l'assembleur et le compilateur C de Digital Research (pack de développement commercialisé par Atari France). Les modifications de dernière heure apportées au GEM ont été prises en compte et un désassemblage complet du système d'exploitation nous permet de fournir ici quantité de précisions sur les différences rencontrées entre la version 520 ST du GEM et celles implementées sur d'autres machines.

Une étude précise des circuits périphériques et spécialement du 6301 dédié aux entrées clavier et souris, ainsi que des cartes détaillées des différentes zones de mémoire, devraient fournir quelque aliment au « bidouilleur » ou, plus noblement, au lecteur désireux d'outrepasser le cadre strict des applications structurellement normées (et par le fait même courantes sinon rebattues). La ligue « A », orgueil du programmeur en assembleur 68000, est disséquée à la lumière de nombreux exemples.

Les processus de compilation, d'assemblage et d'édition de liens, de par leur importance dans toute programmation, sont également analysés et quelques pièges couramment rencontrés lors de ces phases critiques sont explicités. Enfin, dans la mesure où il est possible d'utiliser en Basic les fonctions du système d'exploitation et du GEM, il nous a paru utile de détailler les règles d'utilisation de ces dernières à partir d'un langage certes lent mais connu de l'immense majorité des lecteurs.

Surpris du silence général des auteurs sur la question primordiale des erreurs de programmation, dont la recherche occupe souvent un temps considérable, nous avons mis l'accent, chaque fois que cela paraissait utile, sur les difficultés et les pièges rencontrés à l'utilisation des langages étudiés. Les méthodes de programmation et d'analyse d'erreur, tant en C qu'en assembleur, occupent donc une part non négligeable des pages qui suivent.

TABLE DES MATIÈRES

Introduction VII

1. Généralités 1
   1.1. Organisation matérielle du 520 ST 1
      1.1.1. Le microprocesseur 68000 1
      1.1.2. Système graphique 5
      1.1.3. Sous-système d'entrée-sortie 8
   1.2. Organisation logicielle du 520 ST 12
   1.3. Langages de programmation 13

2. Programmation en langage C 17
   2.1. Le langage C 17
      2.1.1. Premier programme 18
      2.1.2. Les identificateurs 19
      2.1.3. Les constantes 19
      2.1.4. Déclarations et classes d'allocation 20
      2.1.5. Les opérateurs 21
      2.1.6. Les instructions 24
      2.1.7. Directives de compilation 31
      2.1.8. Les tableaux 32
      2.1.9. Les pointeurs 33
      2.1.10. Les structures 34
   2.2. Les bibliothèques C 35
      2.2.1. La bibliothèque standard 35
      2.2.2. La bibliothèque GEM 36
   2.3. Exemple de programme : CALCULATRICE 46
   2.4. Erreurs de programmation 55
      2.4.1. Erreurs de syntaxe 55
      2.4.2. Méconnaissances 56
      2.4.3. Les pièges 57
      2.4.4. Les confusions de bibliothèques 59
      2.4.5. Erreurs de documentation 59

3. Programmation en Assembleur 61
3.1. Méthodes de programmation 61
3.1.1. Instructions sophistiquées 62
3.1.2. Particularités de certaines instructions 66
3.1.3. Utilisations de fonctions externes en assembleur 67
3.1.4. Structuration des programmes 69
3.2. Exemple de programme : DISQUE VIRTUEL 72
3.3. La ligne « A » 80
3.4. Abuser de la ligne « A », user de la ligne « F » 98

4. Liens entre langages 101
4.1. Compilation et édition de liens 101
4.2. Accès aux fonctions GEM en Basic 106
4.3. Mutitâche et accessoires 112
4.3.1. Quel multitâche ? 112
4.3.2. Accessoires 113

5. Programmation des entrées-sorties 117
5.1. Les interruptions 117
5.2. Clavier, souris, horloge et manette de jeu 121
5.3. Programmation du synthétiseur de son 130
5.4. Programmation des entrées-sorties 135

ANNEXES

1. Fonctions C standard 139
2. Fonctions GEM 149
Fonctions VDI 149
Fonctions AES 189

3. Carte Mémoire 227
Variables système 228
Périphériques 231

4. Traps 233
Trap 1 235
Trap 2 249
Trap 13 250
Trap 14 254

5. Jeu d'instructions du 68000 267

BIBLIOGRAPHIE 277

INDEX 279

1. GÉNÉRALITÉS

1.1. Organisation matérielle du 520 ST

L'architecture matérielle du 520 ST peut être décomposée en trois systèmes fonctionnels principaux :

Le cœur, ordonné autour d'un microprocesseur 68000. Pourvu de 16 lignes de données et de 24 lignes d'adresses, l'unité centrale du 68000 travaille sur des opérandes de 32 bits et peut adresser jusqu'à 16 Mo de mémoire. Dans la version de base, 512 Ko de mémoire vive et 192 Ko de mémoire morte résident dans le 520 ST. Le microprocesseur peut par ailleurs accéder sans intermédiaire (DMA ou Direct Memory Access) à une cartouche de ROM enfichable dans l'appareil ainsi qu'à un disque dur éventuel.

Le système graphique, lequel utilise 32 Ko de la mémoire vive, chacun des points de l'écran ayant un correspondant direct (un bit en noir et blanc, deux bits en 4 couleurs, quatre bits en 16 couleurs) dans cette mémoire d'écran (technique dite du Bitmap). En polychromie, les 4 ou 16 couleurs disponibles peuvent être choisies parmi une palette de 512 couleurs.

Les différents systèmes d'entrée-sortie : clavier et souris pilotés par un microprocesseur, interface musicale constituée d'un synthétiseur de sons à trois voies et d'une entrée-sortie MIDI, interfaces parallèle et série, interface disquette et disque dur.

1.1.1. Le microprocesseur 68000

Examiner l'interfaçage matériel du 68000 avec l'ensemble des composants du système dépasserait largement les limites de cet ouvrage. Seuls seront

examinées ici les caractéristiques de ce microprocesseur directement utiles et nécessaires pour la programmation en assembleur.

Vrai 16 bits, c'est-à-dire effectuant les lectures et écritures de données sur seize lignes, le 68000 dispose de possibilités d'adressage étendues et surtout peut effectuer ses opérations internes sur 32 bits. Cadencé par un quartz à 8 MHz, soit 8 millions de cycles machine par seconde, il exécute une addition de deux entiers de 16 bits (donc compris entre – 32768 et + 32767) en 4 cycles machine soit 0.5 microseconde et une multiplication de deux entiers en 70 cycles maximum soit 8.75 microsecondes.

Comme l'indique la figure en page 3, le 68000 dispose de :

— 8 registres de données de 32 bits, sur lesquels sont possibles des opérations portant sur un octet (8 bits), un mot (16 bits) ou un long mot (32 bits).

— 7 registres d'adresse de 32 bits (A0-A6) et un pointeur de pile (A7). Ces registres servent habituellement de pointeurs ou d'index mais peuvent aussi être utilisés comme registres de calcul. Le pointeur de pile constitue en fait un double registre, comme il est expliqué plus bas.

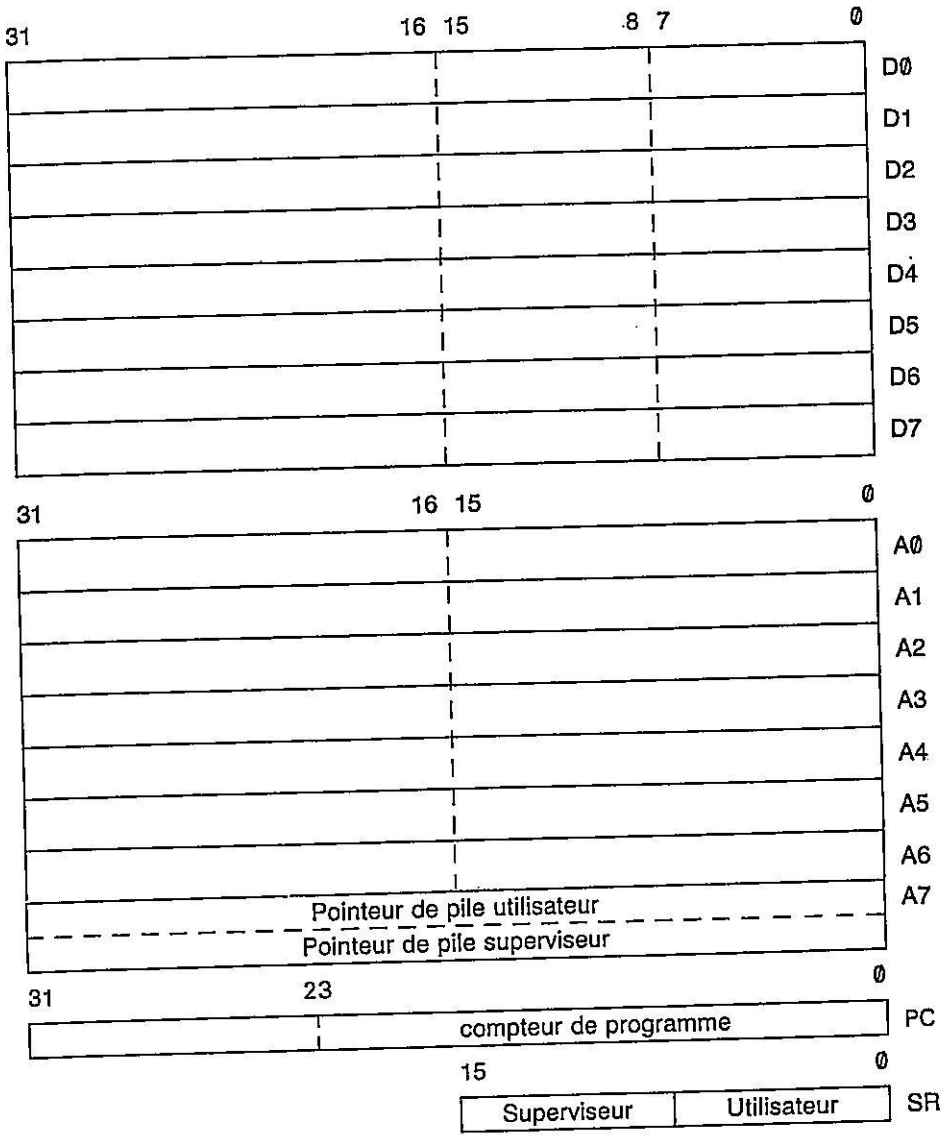
— 1 compteur de programme (PC), dont seuls les 24 bits inférieurs sont utilisés. Ce registre pointe en permanence l'adresse de l'instruction exécutée par le microprocesseur. Il est son repère.

— 1 registre d'état (SR) de 16 bits. Ce registre indique les conséquences de l'exécution des instructions au fur et à mesure de celle-ci (retenue, résultat nul, etc.) ainsi que les états du microprocesseur (interruption, mode).

Tous ces registres font partie intégrante du microprocesseur et n'ont donc naturellement pas d'adresse mémoire. Ils forment la zone de travail du 68000 et sont, par conséquent, très rapidement accessibles. Ainsi, alors que le report du contenu d'un registre de données de 32 bits dans un autre registre de données est opéré en 4 cycles machine (soit 0,5 microseconde), le report du contenu d'une mémoire de 32 bits dans une autre mémoire nécessite 28 cycles (soit 3.5 microsecondes).

Les registres d'adresses, s'ils constituent la source d'une instruction, peuvent être employés comme une zone de 16 ou de 32 bits, selon la taille précisée par l'instruction mais, si l'un de ces registres constitue la destination, il est *entièrement concerné*, c'est-à-dire que les 32 bits du registre sont affectés. Si le bit fort du mot placé dans le registre d'adresse est nul, alors le mot fort du registre d'adresse est mis à $0000, sinon le mot fort est mis à $ffff.

A la différence de certains microprocesseurs courants comme le Z80 ou la famille 8086, le 68000 manipule les adresses dans leur sens « logique » ! Ainsi, si l'on place l'adresse $0006 75F3 dans le registre d'adresse A0, la partie haute 00 de l'adresse sera placée dans l'octet du haut (fort) puis 06 sera placé dans l'octet suivant, et 75 et F3 dans les deux suivants.



Contrairement également aux microprocesseurs pré-cités, une instruction du 68000 comporte toujours le code de la source avant celui de la destination. Ainsi « move D1, (A2) » met le contenu du registre de données D1 dans la mémoire dont l'adresse est contenue dans le registre d'adresse A2.

L'une des spécificités du 68000 est constituée par le double mode de fonctionnement possible : mode SUPERVISEUR et mode UTILISATEUR. Cette distinction, initialement prévue pour faciliter le multi-utilisateur, est prise

en compte par le 520 ST : la partie basse de la mémoire ne peut être lue ou écrite qu'en mode superviseur (adresses 0 à $7FF), le reste étant accessible dans les deux modes. Cette protection des mémoires basses, lesquelles contiennent les variables système, fera l'objet de développements ultérieurs (cf. chapitre 3).

En dehors de cette distinction des espaces mémoire adressables, le passage de mode superviseur en mode utilisateur, et vice-versa, a les caractéristiques suivantes :

Mode SUPERVISEUR

Conditions d'accès : toute interruption matérielle ou logicielle.
Test de mode : bit 13 du registre d'état SR à 1.
Instructions interdites : aucune.
Registres accessibles : D0 à D7, A0 à A6 et pointeur de pile superviseur.
Masquage des interruptions : possible.
Mode TRACE : possible.

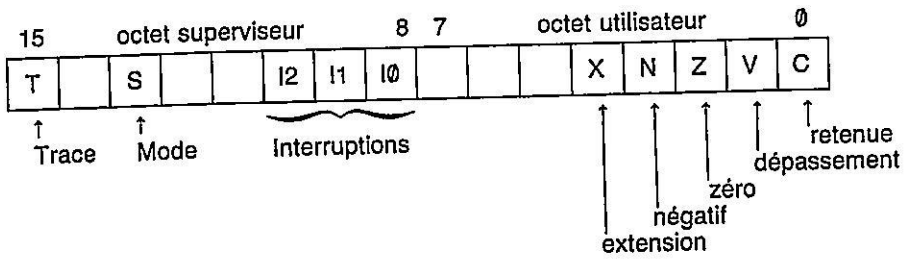
Mode UTILISATEUR

Conditions d'accès : après une sortie du superviseur par RTE ou par la mise à zéro du bit 13 du registre d'état SR.
Test de mode : bit 13 du registre d'état SR à 0.
Instructions interdites : dites « privilégiées », ce sont :

STOP	: mise en attente du microprocesseur
RESET	: validation de la ligne Reset
RTE	: retour d'interruption
MOVE.W <donnee>, SR	: place un mot dans le registre d'état
ANDI.W <donnee>, SR	: ET logique avec le registre d'état
EORI.W <donnee>, SR	: OU exclusif avec le registre d'état
ORI.W <donnee>, SR	: OU logique avec le registre d'état
MOVEA.L An, USP	: charge une adresse dans le pointeur de pile utilisateur
MOVEA.L USP, An	: stocke le pointeur de pile utilisateur dans un registre d'adresse

Registres accessibles : D0 à D7, A0 à A6 et pointeur de pile utilisateur.
Masquage des interruptions : impossible.
Mode TRACE : impossible.

Le registre d'état, comme on le constate, est fortement affecté par le mode de fonctionnement du 68000. Son bit 13 reflète ce mode et l'octet fort de ce registre ne peut être modifié en mode utilisateur. Ce registre SR a la structure suivante :



Les bits 5 à 7, 11, 12 et 14 du registre d'état sont inutilisés et toujours nuls. Seuls deux niveaux d'interruptions (niveaux 2 et 4) sont utilisés sur le 520 ST. Ils correspondent à la mise à 1 du bit 9 et du bit 10 du registre d'état, respectivement. Ces interruptions seront étudiées au chapitre 5.

Le jeu d'instruction du 68000 et son utilisation feront l'objet du chapitre 3 et de l'Annexe 5.

1.1.2. Système graphique

Avec un moniteur monochrome, la résolution graphique du 520 ST est de 640 * 400 points, soit 256 000 points. Chacun de ceux-ci peut être noir ou blanc. Pour conserver en mémoire la valeur de ces points, il faut donc 256 000 bits, soit 256 000/8 = 32 000 octets. Ces 32 000 octets constituent la mémoire d'écran, laquelle est stockée dans la partie haute de la mémoire vive, soit de $78000 à $7FCFF dans la version de base du 520 ST.

On notera à ce propos que 32 000 octets ne font pas 32 Ko, puisqu'un Ko est égal à 2^10 octets, soit 1024 octets. Le lecteur attentif pourra s'interroger sur l'utilisation des 768 octets restants (de $7FCFF à $7FFFF). Avec raison... car ils sont inutilisés ! Pour un motif demeuré obscur, l'initialisation du système aligne le début de la mémoire d'écran sur un compte entier de Ko, donc inéluctablement en $78000 !

Aux 640 points horizontaux de l'écran, dans une résolution monochrome, correspondent 80 octets. Chacune des lignes de l'écran est donc traduite par 80 octets successifs. Si le premier octet de la mémoire d'écran, d'adresse $78000, est égal à $62, soit 0110 0010 en binaire, cela signifiera que les deuxième, troisième et septième points de la première ligne de l'écran sont noirs, etc.

Avec un moniteur polychrome (ou une télévision, via la prise Peritel), deux résolutions sont disponibles :

— la moyenne résolution, 640 * 200 points, soit 128 000 points pouvant prendre 4 couleurs.

— la basse résolution, 320 * 200 points, soit 64 000 points pouvant prendre 16 couleurs.

Aux 128 000 points de la moyenne résolution correspondent toujours 32 000 octets car un point nécessite deux bits pour le codage de la couleur :

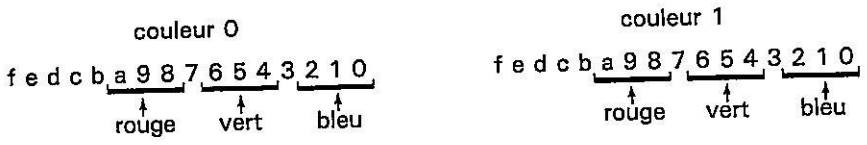
00 code de la couleur blanche
01 code de la couleur rouge
10 code de la couleur verte
11 code de la couleur noire

Aux 64 000 points de la basse résolution correspondent aussi 32 000 octets, un point nécessitant quatre bits pour le codage de la couleur :

0000 code de la couleur blanche	($777)
0001 code de la couleur rouge	($700)
0010 code de la couleur verte	($070)
0011 code de la couleur jaune	($770)
0100 code de la couleur bleue	($007)
0101 code de la couleur magenta	($707)
0110 code de la couleur cyan	($077)
0111 code de la couleur blanc cassé	($555)
1000 code de la couleur grise	($333)
1001 code de la couleur rose	($733)
1010 code de la couleur vert clair	($373)
1011 code de la couleur jaune clair	($773)
1100 code de la couleur bleue claire	($337)
1101 code de la couleur magenta clair	($737)
1110 code de la couleur cyan clair	($377)
1111 code de la couleur noire	($000)

On notera que ce nuancier ne respecte pas vraiment la hiérarchie des couleurs ni même les harmonies de tons. L'électronique a ses raisons... Les valeurs hexadécimales entre parenthèses correspondent aux valeurs de ces couleurs dans la palette de base du 520 ST.

Pour envoyer les signaux vidéo correspondant à chacune des 4 ou des 16 couleurs utilisées en moyenne et basse résolution, le système consulte en effet une palette (stockée en mémoire), laquelle fournit pour chacune des couleurs le pourcentage de rouge, de vert et de bleu de cette couleur. Ce pourcentage est codé sur 3 bits (ce qui offre 8 possibilités par couleur fondamentale, donc un éventail possible de 8*8*8 soit 512 couleurs composites). En mémoire, la palette est codée ainsi :



Ainsi la couleur blanche, de code $777, correspond à une saturation de rouge, de vert et de bleu (100 %), tandis que le magenta clair correspond

à 100 % de rouge, 37,5 % de vert et 100 % de bleu. Les pourcentages correspondent naturellement à des intensités chromatiques et non à des dosages proportionnels ! Il est loisible au programmeur de modifier ces pourcentages, donc les couleurs de la palette, par le biais des instructions Setpallete (a) et Setcolor (a.b) (cf. Annexe 4. Traps).

La haute résolution est dite mono-plan car il y a correspondance linéaire entre les points de l'écran et les bits de la mémoire d'écran : à deux points contigus d'une même ligne correspondent deux bits qui se suivent. A contrario, la moyenne et la basse résolution sont dites multi-plans car les bits de couleur d'un même point ne sont pas contigus en mémoire écran. Cette mémoire est découpée en mots (de 16 bits) selon le principe suivant :

4 couleurs 2 plans	plan 0	plan 1	plan 0	plan 1	...
16 couleurs 4 plans	plan 0	plan 1	plan 2	plan 3	...

Ainsi le premier point en haut à gauche de l'écran aura son premier bit de couleur codé dans le bit fort de $78000 (début de la mémoire écran) et son second bit de couleur codé dans le bit fort de $78002 (un mot plus loin), en mode moyenne résolution. En basse résolution son troisième bit de couleur correspondra au bit fort de $78004 et son quatrième au bit fort de $78006.

Cette structure mono ou multi-plan est d'une importance capitale pour la bonne utilisation des fonctions GEM, spécialement avec un moniteur polychrome. A titre indicatif, la structure multi-plan, décomposant l'écran en unités factuelles de 16 points, est à la source du traitement graphique par entités horizontales de 16 points, lequel se traduit par exemple par des décalages horizontaux des fenêtres toujours égaux à des multiples de 16 points... Par contre les déplacements verticaux peuvent être effectués sur un décalage d'un point seulement. Il n'y a naturellement aucun obstacle théorique à un décalage horizontal inférieur à 16 points mais les problèmes de gestion de la mémoire écran en résultant sont sévères !

Une résolution intermédiaire entre la moyenne et la basse résolution est théoriquement possible et prévue dans les instructions GEM. Elle autoriserait 8 couleurs avec un codage multi-plan sur 3 plans. Malheureusement cette possibilité n'a pas été implementée dans le système d'exploitation du 520 ST. Il est vrai que 640 n'est pas divisible par 3 !

1.1.3. Sous-systèmes d'entrée-sortie

Entrées-sorties clavier

L'expression « entrées-sorties » en matière de clavier peut surprendre, mais le clavier étant contrôlé par un microprocesseur 6301, il est effectivement possible d'envoyer des commandes vers le clavier, par exemple pour une reconfiguration (cf. exemple du chapitre 5).

Le microprocesseur 6301 est un circuit peu connu puisqu'utilisé seulement pour des automates industriels et un micro-ordinateur : le portable HX-20 d'Epson. Il s'agit d'une version C-MOS, légèrement modifiée, du 6801, lui-même dérivé du fameux 6800 de Motorola.

Le 6301 est un véritable micro-ordinateur puisqu'il contient une RAM de 128 octets, une ROM de 4 Ko, une interface série rapide et un programmateur intégrés. Cadencé par un quartz à 4 Mhz sur le 520 ST. le 6301 exécute un million de cycles par seconde. L'interface série rapide fonctionne à une vitesse de 7 812.5 bits/seconde.

Les fonctions du 6301 sont multiples :

— contrôle de l'état des touches du clavier
— contrôle de la position de la souris
— contrôle des boutons de la souris
— contrôle de la (ou des) manette(s) de jeu (position du manche et bouton de mise à feu)
— gestion de l'horloge du 520 ST (date et heure)

Disposant d'un compteur de cycles (sur 16 bits) incrémenté tous les millionièmes de secondes, le 6301 peut également faire office de base de temps pour des temporisations très fines. Cette possibilité et de nombreuses autres, plus classiques, feront l'objet d'une analyse détaillée au chapitre 5.

Synthétiseur de son

Le générateur de son programmable YM 2149 dont est doté le 520 ST est sans doute l'un des circuits les moins utilisés... puisqu'aucune des applications développées jusqu'ici n'a, à notre connaissance, utilisé ses possibilités sinon pour un vague bruitage répétitif ! Il est vrai que la documentation le concernant est plus que restreinte et que seule une instruction du système d'exploitation autorise son emploi...

Ce circuit offre pourtant des ressources étendues puisqu'il comporte trois canaux de sortie programmables indépendants, un générateur de bruit blanc mixable séparément avec chacun des canaux et un générateur d'enveloppe doublé d'un contrôle de l'amplitude de l'enveloppe. Utilisant une fréquence

d'horloge de 2 MHz, la base de temps est divisée par 16 pour la génération de l'onde carrée de chacun des canaux, d'où une valeur de la période de son de 8 microsecondes, ce qui autorise la sortie de son compris entre 30.5 Hz et 125 KHz.

16 registres de base de 8 bits sont disponibles, dont 2 constituent des ports d'entrées sorties sans rapport avec la production de sons. Ces registres ont la fonction suivante :

Registres 0 et 1 : les bits 0 à 3 du registre 1 constituent les bits *forts* de la fréquence de sortie du canal A et les bits 0 à 7 du registre 0 constituent les bits faibles de cette fréquence de sortie. Les bits 4 à 7 du registre 1 sont inutilisés. La fréquence est donc calculée sur 12 bits, selon le schéma :

registre 1	registre 0
b3 b2 b1 b0	b7 b6 b5 b4 b3 b2 b1 b0

Exemple : si registre 1 = $3 et registre 0 = $40, on aura compteur = ($3 * 256) + ($40) = 768 + 64 = 832
soit une période de : 832 * 8 = 6 656 microsecondes, correspondant donc à une fréquence de 150.24 Hz

Registres 2 et 3 : identiques aux registres 0 et 1 pour le canal B.
Registres 4 et 5 : identiques aux registres 0 et 1 pour le canal C.
Registre 6 : Générateur de bruit.
Les 5 bits faibles (0 à 4) de ce registre fixent la fréquence du générateur, laquelle peut varier de 977,5 Hz à 125 Khz.
Registre 7 : validation et mixage des registres précédents
bit 0 : valide le ton du canal A si égal à 0
bit 1 : valide le ton du canal B si égal à 0
bit 2 : valide le ton du canal C si égal à 0
bit 3 : mixe le générateur de bruit avec le canal A si égal à 0
bit 4 : mixe le générateur de bruit avec le canal B si égal à 0
bit 5 : mixe le générateur de bruit avec le canal C si égal à 0
bit 6 : si nul, port A en entrée
si égal à 1, port A en sortie
bit 7 : si nul, port B en entrée
si égal à 1, port B en sortie
Registre 8 : registre d'amplitude du signal du canal A.
Seuls les 5 bits faibles (0 à 4) sont utilisés.
Le bit 4 détermine le mode. S'il est nul, l'amplitude est fixe, sinon elle est contrôlée par le générateur d'enveloppe. Les bits 0 à 3 permettent de définir 16 niveaux d'amplitude.
Registre 9 : identique au registre 8 pour le canal B
Registre 10 : identique au registre 8 pour le canal C

Registres 11 et 12 : période de l'enveloppe

La fréquence de l'enveloppe est déterminée selon le principe suivant : les 8 bits du registre 12 sont les 8 bits *forts* de la période et les 8 bits du registre 13 sont les 8 bits faibles. Si "t" désigne le contenu des registres 11 et 12, la période est de :

t * 128 microsecondes soit une fréquence de :

1000000 / (t*128) Hz

Registre 13 : Forme de l'enveloppe

Celle-ci est définie par les 4 bits faibles du registre :

bit 0 : retenue
bit 1 : alternance
bit 2 : attaque
bit 3 : continuité

Registre 14 : Port d'entrée-sortie A.

bit 0 : sélection face disquette (0 = face 1)
bit 1 : sélection du lecteur de disquette A si à 1
bit 2 : sélection du lecteur de disquette B si à 1
bit 3 : signal RTS pour interface série RS 232C
bit 4 : signal DTR pour interface série RS 232C
bit 5 : signal d'échantillonnage sortie parallèle
bit 6 et 7 : inutilisés

Registre 15 : Port d'entrée-sortie B.

port de sortie parallèle (sur 8 bits).

La programmation de ces registres, illustrée de quelques exemples, fera l'objet d'un développement au chapitre 5.

Interfaces parallèle et série

Les registres de contrôle et de sortie de l'interface parallèle correspondent au bit 5 du port A et au port B du circuit générateur de son étudié ci-dessus. L'entrée « Busy » (demande par l'imprimante d'interruption d'envoi des données) est par contre reliée à la ligne d'interruption 10 du circuit périphérique 68901, lequel contrôle également les lignes d'entrée-sortie de l'interface série RS 232C. Une étude précise de ce circuit sophistiqué dépasserait le cadre de cet ouvrage, aussi le lecteur est-il invité à se reporter au livre de P. Jaulent (« Circuits Périphériques de la famille 68000 », éd. Eyrolles, 1985).

La vitesse de transmission série est initialisée à partir des registres de contrôle et de donnée du timer D du 68901. Le signal d'horloge de 76,8 Khz fournit la vitesse série après division selon le contenu des registres pré-cités. Les valeurs de division pour le registre de contrôle TCDCR sont les suivantes :

bit 2	bit 1	bit 0	division
0	0	1	par 4
0	1	0	par 10
0	1	1	par 16
1	0	0	par 50
1	0	1	par 64
1	1	0	par 100
1	1	1	par 200

La fréquence obtenue est ensuite divisée par le contenu du registre de données du timer D.

Exemple : si bits 0 à 2 de TCDCR = 001
et registre de données = $C (12 décimal)
la fréquence sera : (76800/4)/12 = 1600 bits/sec.

Il est ainsi possible de programmer des vitesses d'entrée-sortie peu normalisées. Par contre certaines vitesses classiques sont ici approximatives. Ainsi la vitesse de 3600 bits/s est programmée avec 1 dans TCDCR et 5 dans TDDR (registre de données du timer D), ce qui correspond en fait à une fréquence de 3840 bits/seconde. Le tableau ci-dessous fournit les valeurs respectives à employer dans TCDCR et TDDR pour quelques vitesses courantes :

	TCDCR	TDDR		TCDCR	TDDR		TCDCR	TDDR
19200	1	1	960	1	$14	150	1	$80
9600	1	2	800	1	$18	120	1	$A0
6400	1	3	600	1	$20	100	1	$C0
4800	1	4	480	1	$28	75	3	$40
3200	1	6	400	1	$30	60	3	$50
2400	1	8	300	1	$40	56.9	3	$54
1920	1	$ A	240	1	$50	50	3	$60
1200	1	$10	200	1	$60	45.5	2	$A9

Les trois dernières vitesses correspondent à des fréquences RTTY...

Autres entrées-sorties

Une entrée et une sortie MIDI sont destinées à la communication avec un synthétiseur. Elles sont considérées par le 520 ST comme une entrée et une sortie série, dont la vitesse est fixée à 31 250 bits/seconde.

L'interfacage des lecteurs de disquettes s'opère par le biais d'un circuit WD 1770 tandis que l'interfacage du disque dur s'effectue via un contrôleur de DMA (Accès direct à la mémoire).

Les adresses mémoires des variables système liées aux entrées-sorties sont fournies en Annexe 3. Nous reviendrons par ailleurs, dans le chapitre 5, sur l'utilisation des informations qui précèdent, à travers quelques exemples de manipulation directe des entrées-sorties.

1.2. Organisation logicielle du 520 ST

Pour relier l'ensemble des composants matériels précédemment décrits et pour permettre à l'utilisateur d'en profiter pleinement, un système d'exploitation de base est indispensable. Appelé TOS (pour Tramiel Operating System), il regroupe l'ensemble des routines de démarrage du système, de gestion de la mémoire et de gestion des entrées-sorties.

Naturellement écrit en assembleur, ce système est fortement inspiré dans sa structure du CP/M 68K, c'est-à-dire de la version du CP/M pour microprocesseur 68000. La distinction entre le BDOS (système d'exploitation disque) et le BIOS (gestion des entrées-sorties) est bien effectuée, les primitives de base du CP/M se retrouvent. Mais il serait bien peu sérieux d'affirmer que le 520 ST possède un système d'exploitation CP/M :

1) le CCP (programme de contrôle) est inexistant et n'est disponible que dans le système de développement commercialisé séparément. Certaines primitives essentielles du CCP telle que la commande PIP ne sont d'ailleurs pas implementées dans ce programme...

2) l'ensemble des exceptions (Trap 2) qui servent en CP/M 68k à l'appel des primitives système est absent du TOS.

Se permettant une réflexion tout à fait subjective, l'auteur peut affirmer qu'il n'y a rien là de dramatique, le CP/M étant relativement lourd d'emploi et inadapté aux 16 bits, surtout lorsque ceux-ci disposent du GEM et d'une souris ! Quel intérêt aurait la commande PIP quand un simple cliquage et déplacement de souris permet de copier un fichier ?

Car le TOS n'est que la première couche du système d'exploitation, si l'on considère celui-ci dans le sens large de l'ensemble des routines système. Une deuxième couche, dite VDI (« Virtual Device Interface »), regroupe l'ensemble des fonctions graphiques tandis qu'une troisième couche, dite AES (« Application Environment Services »), comprend l'ensemble des routines de gestion des fenêtres, de manipulation de la souris, de déroulement des menus, etc. Le VDI et l'AES forment les deux volets du GEM (« General Environment Manager »), ils sont toutefois dissociables et il existe même une instruction permettant d'éliminer l'AES de la mémoire, tant que celui-ci est en mémoire vive naturellement !

A titre indicatif, l'ensemble des routines d'entrée-sortie et de gestion des disquettes occupe environ 16 Ko dans le système fourni en janvier 86 (non compris les routines de démarrage du système et les zones de variables), alors que le VDI occupe environ 56 Ko et l'AES 96 Ko (non compris également les zones de variables). Il est vrai que le VDI comme l'AES ont été écrits

en langage C, ce qui ne joue pas en faveur de la concision du fichier objet produit...

Mises à part les routines de démarrage du système et de traitement des interruptions, l'ensemble des routines *du TOS et du GEM* sont disponibles pour le programmeur *en C comme en assembleur*. Nombre de ces routines sont également utilisables en Basic via les instructions VDISYS et GEMSYS.

La procédure d'appel est naturellement différente en assembleur et en C bien que l'utilisation de macros contribue à rapprocher fortement ces deux langages sur ce point précis. Cette procédure, caractéristique du 68000, utilise les exceptions (« Trap »), autrement dit des interruptions logicielles. A l'appel d'un Trap, le microprocesseur passe en mode superviseur, place dans le compteur de programme l'adresse du vecteur de Trap et exécute la routine débutant à cette adresse puis, après exécution, repasse en mode utilisateur.

Le 68000 autorise 16 niveaux d'exceptions, numérotés Trap 0 à Trap 15. Sur le 520 ST, seuls 4 de ces niveaux sont utilisés par le système d'exploitation (les autres étant disponibles à l'utilisateur). Ce sont :

TRAP 1 : ensemble des routines BDOS (gestion disques et système)
TRAP 2 : ensemble des routines GEM (VDI et AES)
TRAP 13 : primitives d'entrée-sortie (BIOS)
TRAP 14 : primitives supplémentaires d'entrée-sortie (TOS)

La sélection de telle ou telle routine dans un même Trap est effectuée en fonction des paramètres empilés avant l'appel, ou du contenu du registre de données D0 pour Trap 2. L'utilisation de ces Traps fait l'objet d'exemples au chapitre 3 et une analyse exhaustive en est faite en Annexe 4.

1.3. Langages de programmation

A la rédaction de ces lignes étaient déjà disponibles pour le 520 ST :

— Le Basic d'Atari, très complet bien qu'encore imparfait.
— Le Logo d'Atari, version également très complète du Dr Logo.
— Deux compilateurs de langage C, deux autres étant annoncés.
— Trois assembleurs 68000.
— Un compilateur Forth, deux autres étant annoncés.
— Un compilateur Modula 2.

Étaient d'autre part en cours d'adaptation deux compilateurs Pascal et un compilateur Lisp, sans parler d'autres projets inconnus de l'auteur... Le moins que l'on puisse dire est que le choix est large ! Aussi, pour ne pas trans-

former ce livre en encyclopédie, a-t-il fallu choisir de n'étudier ici que la programmation en langage C et en assembleur, sans toutefois écarter quelques références au Basic. Ces langages étant les plus utilisés, donc les mieux connus, par l'auteur, il y a certes un parti-pris évident mais qui envisagerait de développer un programme de quelque importance en Logo ? Quant au Forth et à Modula 2, leur diffusion publique demeure marginale. Enfin, les avantages du Pascal pèsent moins lourd que ses inconvénients face au langage C.

La querelle des partisans de l'assembleur et des fanatiques du C est passée de mode tant la race des programmeurs en assembleur semble en voie de disparition ! Il est hors de question de trancher ce débat par quelques formules, mais la lecture de cet ouvrage devrait permettre de mieux cerner les avantages et les inconvénients de chacun de ces langages.

On relèvera que les manipulations fines des entrées-sorties, surtout lorsqu'elles sont peu orthodoxes, demeurent la chasse gardée de l'assembleur. A titre d'exemple, il n'est pas prévu dans les fonctions graphiques disponibles en C d'afficher un caractère d'une couleur avec un fond de caractère différent du fond d'écran. Reconnaissons cependant qu'il s'agit d'un cas de figure dont la pratique du programmeur n'est pas forcément émaillée !

Aussi parfaite soit-elle, la compilation d'un langage ne saurait, quoi qu'il en soit, produire un fichier objet aussi rapide d'exécution qu'un programme écrit directement en assembleur, ne serait-ce qu'à cause du traitement général devant être opéré par l'assembleur. Le traitement d'une boucle vide en assembleur et en langage C en est une illustration :

Version assembleur :

```
.text
_main
            move.w   #9999,d0        8 cycles machine
boucle
            dbra     d0,boucle       10 cycles * 10000
            rts                      16 cycles machine
                                  ---------------------
.end                    total :  100024 cycles
```

Version normale en langage C :

```
main()
{
            int     i;
            for (i=9999;i>=0;i--);
}
```

Cette version une fois compilée produit le code suivant :

```
_main
        link      a6,#$fffa              16 cycles
        move.w    #$270F,$fffe(a6)       16 cycles
        bra       _test                  10 cycles
_boucle
        subq.w    #$1,$fffe(a6)          16 cycles * 10000
_test
        tst.w     $fffe(a6)              12 cycles * 10000
        bge       _boucle                10 cycles * 10000 + 12
        unlk      a6                     12 cycles
        rts                              16 cycles
                                     ---------------
                                     380094 cycles
```

Une version C plus concise peut être écrite, qui utilise un registre comme compteur de boucles :

```
main()
{
        register   i;
        for (i=9999;i>=0;i--);
}
```

Cette version, une fois compilée, fournit le code suivant :

```
_main
        link      a6,#$0                 16 cycles
        movem.l   d6-d7,-(a7)            24 cycles
        move.w    #$270F,d7              16 cycles
        bra       _test                  10 cycles
_boucle
        subq.w    #$1,d7                  4 cycles * 10000
_test
        tst.w     d7                      4 cycles * 10001
        bge       _boucle                10 cycles * 10000 + 12
        movem.l   (a7)+,d6-d7            28 cycles
        unlk      a6                     12 cycles
        rts                              16 cycles
                                     ---------------
                                     180138 cycles
```

On constate qu'une boucle vide de 10 000 pas est exécutée en 12,5 ms si elle est écrite en assembleur contr 47,5 ms en langage C. Il est vrai que la deuxième version en C est exécutée en 22,5 ms, mais l'utilisation d'une variable de type register est rare en C et non naturelle alors qu'elle est la règle en assembleur. Le lecteur expérimenté aura sans doute noté la redondance du fichier compilé. Ainsi le « Link » et l'« unlk » sont inutiles dans la deuxième version, de même que l'empilement du registre D6. Surtout, dans

les deux versions, l'instruction « tst.w » est superfétatoire puisque le « subq.w » qui précède (de même que le move.w de début) positionne les drapeaux du registre d'état !

A titre comparatif, la boucle Basic (« for i = 0 to 10000: next i ») est exécutée en 8,5 secondes par le Basic Atari, performance bien supérieure à nombre de Basic mais naturellement hors de proportion d'avec les précédents langages.

La comparaison précédente, volontairement simpliste bien que nécessitant quelques connaissances de base en C et assembleur 68000 pour être appréciée à sa juste valeur, offre un autre intérêt que les mesures de « timing ». Elle permet d'appréhender la différence de taille d'un code objet assembleur et compilé. Cette différence est d'autant plus nette que les routines compilées sont décomposées en modules élémentaires et que les variables utilisées ne sont pas du type « register »

Il demeure que la programmation en langage C est moins astreignante et plus rapide que la programmation en assembleur et la moindre rapidité n'est pas un inconvénient pour la majeure partie des programmes, surtout avec un microprocesseur comme le 68000. Enfin, la génération d'un code objet plus important n'est pas un réel inconvénient lorsque la mémoire de base est de 512 Ko !

Avant d'aborder ces deux langages, il paraît utile de souligner que, si la connaissance des instructions de base du C est indispensable pour programmer dans ce langage (de même que la connaissance des instructions du 68000 pour l'assembleur), cette connaissance n'est en aucun cas suffisante. Le niveau de sophistication du GEM et les nombreuses entrées-sorties du 520 ST s'accommodent fort mal d'une programmation n'utilisant ni les fonctions GEM ni les fonctions système. L'idée de départ de cet ouvrage résulte d'ailleurs de ce constat : tout manuel de programmation générale en C ou en assembleur est presque caduc face à une machine comme le 520 ST (essayez un exemple tiré d'un de ces ouvrages...). D'autre part un simple manuel de référence fournissant, par exemple, les instructions GEM ou les adresses système, est d'utilisation difficile, surtout pour un programmeur amateur, et passe trop souvent à côté des problèmes pratiques rencontrés en programmation.

2. PROGRAMMATION EN LANGAGE C

Inventé dans les années 70 par Dennis RITCHIE pour écrire le système d'exploitation UNIX, le langage C devait rester longtemps l'apanage de la mini-informatique. Avec l'évolution des micro-ordinateurs et l'apparition pour ceux-ci des premiers compilateurs, la diffusion de C dépassa largement le cadre des seuls initiés. Et non sans raison, car ses qualités sont multiples.

Portabilité (ou presque) sur toutes les machines disposant d'un compilateur C, facilité d'apprentissage vu le nombre réduit d'instructions, efficacité dans la rapidité d'exécution des programmes produits, le C a encore de beaux jours devant lui. Le 520 ST devrait d'ailleurs participer à sa promotion puisque deux compilateurs existent déjà (deux autres annoncés) et le succès du GEM, écrit en C et donc particulièrement conçu pour une programmation en ce langage, saura encore en accroître l'intérêt.

Chaque médaille ayant son revers, le C a ses faiblesses : entre autres, sa lenteur de compilation et son manque d'analyse syntaxique. Ces défauts peuvent toutefois être minimisés par l'utilisation d'un disque virtuel (voir chapitre 3) et d'un programme de « lint ».

2.1 Le langage C

L'auteur n'a pas la prétention de croire que la lecture de ce chapitre dévoilera toutes les richesses du C, mais espère simplement que la présentation rapide de ce langage permettra aux lecteurs débutants de s'en faire une première idée, et aux programmeurs plus avancés de disposer d'un aide-mémoire toujours disponible.

2.1.1. Premier programme

Il ne saurait être question de se soustraire à la tradition qui consiste à présenter le programme capable d'afficher à l'écran le célèbre « COUCOU ».

```
 /* Premier programme */

main ()
{
int i;

 printf ("COUCOU");
 for (i = 0; i < 32000; i++) ;

 }
```

La première ligne de ce programme est un commentaire, équivalent au REM du Basic. Toute chaîne encadrée par le symbole d'entrée /* et de sortie */ est ignorée par le compilateur.

La deuxième ligne **main** () correspond au titre de la fonction. Un programme C est constitué d'un nombre variable de sous-programmes (nommés fonctions en C) reconnus par un titre (un identificateur), permettant de les identifier. Un titre (ici **main**) est suivi de parenthèses ouvrante et fermante qui encadrent les valeurs (on parle d'arguments) que va recevoir la fonction. Même quand celle-ci ne reçoit pas d'argument (c'est le cas ici), les parenthèses demeurent obligatoires.

Le programmeur est libre de donner à une fonction le titre qu'il désire (en respectant cependant quelques règles) mais un programme doit toujours comprendre une fonction (et une seule) s'intitulant **main**. Ce titre indique au compilateur le point d'entrée du programme, c'est-à-dire la première fonction à exécuter.

Le « corps » d'une fonction (l'ensemble des instructions exécutées par une fonction particulière) est délimitée par une accolade ouvrante qui en précise le commencement, et fermante pour stipuler la clôture de la fonction.

La troisième ligne **int i** ; consiste en une déclaration de la variable « i » utilisée dans le programme. Le C a besoin de « reconnaître » tous les objets qu'il utilise. Voir chapitre 2.1.4.

La quatrième ligne **printf** (« **COUCOU** ») n'est pas réellement une instruction C, ce langage ne disposant en effet d'aucune entrée-sortie. C'est un appel à une fonction qui appartient à la bibliothèque C standard, à qui l'on fournit comme argument une chaîne de caractères.

Lors de l'édition de liens, cette fonction sera reconnue et transformée en une routine adéquate. La connaissance des fonctions appartenant aux

bibliothèques est primordiale afin de supplanter les manques du C en ce qui concerne la gestion des périphériques et de faciliter le travail de programmation en évitant d'écrire ce qui existe déjà.

La dernière ligne enfin est une véritable instruction qui bouclera 32 000 fois permettant la temporisation d'affichage.

2.1.2. Les identificateurs

Un identificateur est un nom générique qui permet d'identifier les différents objets manipulés par C, comme les fonctions, les variables, les constantes, etc... Ces identificateurs obéissent à quelques règles syntaxiques qu'il ne faut pas transgresser :

— Un nom peut être composé d'un nombre quelconque de caractères pris parmi les lettres et les chiffres, le premier caractère étant obligatoirement une lettre. Mais il ne peut comprendre ni espaces, ni caractères spéciaux à l'exception du caractère de soulignement (_).
— Seuls les sept premiers caractères de l'identificateur sont pris en compte par le compilateur de Digital Research.
— Majuscules et minuscules sont permises, mais ne sont pas équivalentes. En C, la règle générale veut que l'on utilise les majuscules pour désigner les constantes, les minuscules étant réservées pour les titres de fonctions et de variables.

```
val_tab    di2a    TRUC    fonc_1    fonction
      sont des identificateurs corrects mais

4500     4ieme     fonc 1      test:5
      ne sont pas valides.
```

2.1.3. Les constantes

Les règles syntaxiques à utiliser pour spécifier les types de constantes sont les suivantes :

— en décimal, notation habituelle : 10 250 1300
— en hexadécimal, le chiffre 0 suivi de la lettre x, ou X : Ox1fff
— en octal, précédé du chiffre 0 : 014 0426
— un réel : 3.125 1.023e+1
— un caractère est placé entre apostrophes : 'A' '*' '4'
— une chaîne est placée entre guillemets : "ceci est une chaîne".

On peut augmenter la taille d'une constante entière en faisant suivre sa valeur de la lettre L indiquant ainsi une valeur longue : Ox1L.

2.1.4. Déclarations et classes d'allocation

Une variable est caractérisée par son identificateur (le nom de la variable), son type (sa taille) et sa classe d'allocation (son espace mémoire).

Les types : Il existe cinq types usuels :

* le type char (type caractère : 8 bits)
* le type int (entier machine, soit 16 bits pour le 68000)
* le type long (un long mot : 32 bits)
* le type float (flottant en simple précision : 32 bits)
* le type double (flottant en double précision : 64 bits. N'est pas implémenté sur tous les compilateurs).

La déclaration des variables est obligatoire et doit toujours être terminée par un point-virgule. Le chaînage des déclarations sur une même ligne est possible lorsqu'elles référent à un type identique ; de même qu'il est permis d'initialiser une variable lors de sa déclaration.

```
int x, y, z ;          /* déclare trois entiers */
long val ;             /* déclare une variable de type long */
char carac = 'A' ;     /* déclaration et initialisation */
```

Le qualificatif **unsigned** associé aux types permet de considérer une variable comme une valeur non signée :

```
char c ;               /* valeur de c comprise entre - 128 et + 127 */
unsigned char c ;      /* valeur comprise entre 0 et 225 */
```

La classe d'allocation : Une variable peut être stockée sous différentes formes dans la mémoire (dans la pile, un registre ou une cellule de mémoire permanente). La classe d'allocation spécifie, implicitement ou explicitement, l'emplacement qui lui est alloué.

Les variables déclarées dans le corps d'une fonction (c'est-à-dire après l'accolade ouvrante) sont locales à la fonction et cessent d'exister dès que le programme en sort. Leurs contenus sont perdus. Cela permet de définir des variables qui occuperont la mémoire uniquement lors de leur utilisation et de ne provoquer aucun interférence avec d'autres variables locales qui emploieraient un même identificateur.

Il peut cependant s'avérer nécessaire que le contenu d'une variable locale ne soit pas abandonné entre deux appels à cette fonction. Le mot réservé **static** indique au compilateur de préserver un contenu.

```
fonction ()
{
int  i;                    /*  contenu perdu à chaque sortie de fonction

static  long  compteur; /* le dernier contenu sera préservé


instruction ;
}
```

Les variables déclarées en dehors de toutes fonctions (elles sont généralement déclarées au début du fichier source) sont considérées comme globales et sont reconnues par toutes les fonctions du programme. Utiliser, pour ces variables, un même identificateur provoquerait un message d'erreur du compilateur.

Lorsque une fonction ou l'ensemble d'un programme doit utiliser une variable globale définie dans un autre fichier source, il lui faut spécifier que la variable à employer existe déjà ailleurs. On utilise alors le mot réservé **extern**.

```
int coord_x;            /* variable globale interne */
extern int sauve;       /* variable globale externe au fichier */

main ()
{
int i, j;                /* variables locales à la fonction */

instruction;
}
```

Quand une variable locale doit être fréquemment utilisée (dans une boucle par exemple), la placer dans un registre représentera un gain de rapidité important (cf. chap. 1.3.). Le spécificateur **register** demande au compilateur (si c'est possible) le positionnement de la variable spécifié dans un registre.

register int i ; /* place la variable i dans un registre */

2.1.5. Les opérateurs

Le langage C propose un jeu complet d'opérateurs. Au-delà des « classiques » que l'on retrouve dans de nombreux langages évolués, les opérateurs de manipulation de bits permettent une programmation jusque-là dévolue à l'assembleur.

Les opérateurs arithmétiques :

+ addition
– soustraction

* multiplication
/ division
% reste de la division (utilisable uniquement avec des entiers).

L'opérateur d'affectation :

L'opérateur " = " charge une valeur ou le résultat d'une expression placé à droite de l'opérateur dans une variable située à sa gauche.

Ex : val = 10 x = a * 3 + b res 1 = (z + d) % 2

Il est permis d'enchaîner les affectations.

Ex : a = b = c = d = 0 annule les 4 variables a, b, c, d

Lorsqu'un opérateur se retrouve de part et d'autre du signe " = ", il est possible de simplifier l'écriture.

Ex : a = a + b ou x = x * 2 peuvent s'écrire a += b ; x *= 2 ;

Seuls les opérateurs arithmétiques ou de manipulation de bits permettent ce genre d'écriture.

Les opérateurs de manipulation de bits :

Ces opérateurs ne peuvent s'appliquer aux variables de type flottant.

& ET logique [0 et 0 → 0, 0 et 1 → 0, 1 et 1 → 1]
| OU logique inclusif. [0 et 0 → 0, 0 et 1 → 1, 1 et 1 → 1]
^ OU logique exclusif. [0 et 0 → 0, 1 et 1 → 0, 0 et 1 → 1]
>> Décalage à droite du nombre de bits spécifiés.
Ex : z = z >> 2 (ou z >>= 2) décale z de 2 bits vers la droite. Les bits sortants sont perdus : les bits entrants ont la valeur du bit de signe si la variable est signée, à 0 si unsigned
<< Décalage à gauche du nombre de bits spécifiés.
Ex. v = x << 4 les bits entrants sont mis à 0.
~ Opérateur unaire de complémentation à 1. [0 → 1, 1 → 0]

Les opérateurs d'incrément et de décrément :

Les opérateurs + + et − − provoquent respectivement une incrémentation ou décrémentation de la variable qui leur est associée.

Ex : i+ +, i− − ajoute ou retranche une unité à la variable i.

Ces opérateurs peuvent être placés en préfixe pour que la valeur de la variable soit transformée avant son utilisation.

Ex : si y = 3, x = y+ + → x = 3 puis incrément de y : y = 4
x = + +y → incrément de y, puis affectation : x et y = 4

Bien que ces opérateurs fassent triple emploi (i + + équivalent à i = i + 1, équivalent à i + = 1), leur utilisation est vivement conseillée : le compilateur les optimise au mieux.

Les opérateurs de comparaison :

Ces opérateurs comparent deux expressions et donnent un résultat VRAI (non nul), ou FAUX (= = 0).

<	inférieur
< =	inférieur ou égal
>	supérieur
> =	supérieur ou égal
= =	identique (à ne pas confondre avec l'opérateur "=")
! =	différent

Ex : x = a > b ; x est non nul si a > b, égal à 0 si a < = b.

Les opérateurs logiques :

Une opération logique est effectuée et donne un résultat VRAI (non nul) ou FAUX (= = 0).

&& ET logique : résultat vrai si toutes les expressions vraies.

Ex : res = ((a ! = b) && (a < c) && (c = = e))

:: OU logique : résultat vrai si au moins une expression vraie.

Ex : val = ((x > y) (x > z))

! NÉGATION unaire : un opérande VRAI devient FAUX, et réciproquement.

Ex : x = !y x est VRAI si y est FAUX, FAUX si y est VRAI.

Les opérateurs d'adressage :

Ces opérateurs permettent d'accéder aux objets selon différentes méthodes qui seront explicitées dans les chapitres les concernant.

[]	Permet l'adressage indexé
&	Donne l'adresse d'un objet (le contexte le différenciant de l'opérateur logique de bit)
*	Fournit le contenu d'une adresse (même remarque que précédemment pour l'opérateur de multiplication)
– >	Permet l'accès aux membres d'une structure par une adresse
.	(point) Permet l'accès aux membres d'une structure par une variable

Les divers :

L'opérateur **sizeof** fournit la taille en octets de l'opérande ou du type de base (char, int, etc...) qui lui est associé.

Ex : sizeof (nom d'un tableau de 20 entiers) donnera 40.
sizeof (int) sera égal à 2 (pour un micro 16 bits).

L'opérateur ternaire **? :** est fréquemment rencontré parce que permettant, avec concision, d'opérer un choix entre deux alternatives. Il met en relation trois opérandes sous la forme suivante :
(expression vraie) ? alors expression 1 : sinon expression 2

Ex : x = ((a > b) ? a : b) affecte à x le contenu de la plus grande des deux variables

Priorité des opérateurs :

Le tableau ci-dessous résume l'ordre de priorité des opérateurs. Cette table est classée par ordre décroissant, les opérateurs ayant la précédence équivalente étant placés sur une même ligne. Lors de l'écriture d'un programme, n'hésitez pas à employer les parenthèses si vous voulez être certain de la priorité à accorder aux opérateurs.

```
----------------------------------------------------
 ()  []   ->   .
 !   ~    ++   --   -   (type)   *   &   sizeof   _
 *(mul)   /    %
 +   -
 <<   >>
 <    <=   >   >=
 ==   !=
 & (ET binaire)
 ^
 |
 &&
 ||
 ?:
 =   +=   -=   *=   etc...
----------------------------------------------------
```

2.1.6. Les instructions

Le C prouve, par l'exemple, qu'un nombre restreint d'instructions n'empêche pas la puissance d'un langage. Ces instructions ont été classées en deux groupes :

— les instructions simples.
— les instructions composées.

Si ce classement n'est évidemment pas le seul possible, il va nous permettre d'énoncer quelques règles simples qui deviendront autant de moyens mnémotechniques susceptibles d'éviter certaines erreurs au débutant.

Les instructions simples

Est simple l'instruction qui se suffit à elle-même, par opposition à l'instruction composée qui est suivie, elle, d'une ou plusieurs instructions simples. **Une instruction simple est toujours terminée par un point virgule.**

* *L'instruction d'appel*

Pour clarifier l'ensemble d'un programme, il est toujours préférable d'écrire un grand nombre de fonctions élémentaires plutôt que quelques fonctions de taille importante. L'instruction d'appel de fonctions est donc fréquemment rencontrée.

Elle s'écrit sous la forme :

nom (argument 1, argument 2, ... argument n) ;

où « nom » est l'identificateur (le titre) de la fonction, suivi, s'il en est besoin, de la liste d'arguments à fournir à la fonction appelée.

Les arguments envoyés lors de l'appel d'une fonction sont des variables locales, inconnues de la fonction appelée.

La fonction appelée doit donc déclarer les arguments reçus pour pouvoir les utiliser et travaillera sur les variables provisoires qu'elle s'est créées. *Les variables de la fonction appelante ne seront pas modifiées.* (Nous verrons plus loin qu'il est possible de fournir comme argument l'adresse d'une variable permettant ainsi d'en modifier la valeur).

La déclaration des arguments s'effectue entre l'identificateur et le corps de la fonction.

```
main()      /* le point d'entrée du programme */
{
déclaration des variables locales;

instruction 1;
instruction 2;
ceci (a.b.c);  /* appel de fonction avec 3 arguments */
cela ();       /* appelle la fonction sans arguments   */
}              /* fin du programme et retour           */

ceci (x,y.z)   /* l'instruction reçoit les valeurs des arguments */
déclaration de x.y,z; /* déclaration des arguments              */
{
déclaration;  /* déclaration des variables locales à la fonction */

instruction 1;
instruction 2;
}              /* retour à la fonction appelante       */

cela ()        /* la fonction ne reçoit pas d'argument */
{
déclaration;

instruction 1;
instruction 2;
}              /* retour à la fonction appelante       */
```

* *L'instruction d'affectation*

Cette instruction utilise l'opérateur " = " pour affecter le résultat d'une expression à une variable. L'instruction s'évalue de droite à gauche.

variable = expression ;

ou *expression* peut être :

— une constante

a = b = 0 ; /* annule les variables a et b */

code = "A" ; /* charge dans « code » la valeur ASCII de A (65) */

— une autre variable

val_1 = val_2 ;

— une expression arithmétique, logique ou booléenne

res = a*b+3 ; majus = minus & OXDF ; plus_gr = (a > b) ;

— une valeur renvoyée par une fonction

ret = truc () ; /* la fonction « truc » retourne une valeur qui est placée dans la variable */

Les incrémentations et décrémentations, qui sous-entendent l'opérateur " = " (i = i + 1), peuvent être également considérées comme des instructions d'affectation.

Ex : i++; i--; ++i; --i;

* *L'instruction RETURN*

L'accolade fermante qui termine le corps d'une fonction est toujours traduite par le compilateur comme un retour au programme appelant (RTS en 68000). L'instruction **return** a les mêmes effets mais est employé dans les deux cas suivants :

— à la suite d'un test, le retour à la fonction appelante doit s'opérer avant la fin de la fonction,

— la fonction appelante réclame une valeur en retour.

return (résultat d'une expression) ;

où *expression* peut-être d'un type complexe (arithmétique, booléen, etc...), mais représentant une unique valeur de retour.

```
déclarations des variables globales;

main ()
{
déclaration des variables locales;

instruction 1;
```

```
fait ();
resul = teste (); /* dans "resul", valeur retournée par la fonction */
}

fait ()
{
déclaration des variables locales;

instruction 1;
if (condition) return (); /* si condition remplie, retour immédiat */
instruction 2;
}                         /* sinon retour en fin de fonction       */

teste ()
{
déclaration des variables locales;

instruction 1;
if (condition) return(0); /* si condition remplie, retourne 0 */
return (a+b+c);           /* sinon retourne un résultat algèbrique  */
}
```

Par défaut, une fonction retourne une valeur entière. Si tel n'est pas le cas, elle doit être déclarée en spécifiant le type de la valeur qu'elle renvoie. Le type retourné précéde l'identificateur de fonction lorsque la fonction est interne, sinon la déclaration est explicite.

```
float val_abs (x)      /* fonction retournant un flottant */
float x;               /* déclaration de l'argument reçu  */
{
float y;
float fabs ();  /* déclaration de la fonction externe appartenant
                 à la bibliothèque C retournant une valeur absolue */
 y = fabs (x);  /* y = valeur absolue de x */
 return (y);    /* retourne un flottant */
}
```

* *L'instruction BREAK. L'instruction CONTINUE*

Ces deux instructions, placées à l'intérieur d'une boucle, sont généralement associées à un test de condition. L'instruction **break** oblige un programme à sortir d'une boucle. L'instruction **continue** force la réitération immédiate d'une boucle.

Nous en verrons des exemples concrets lorsque nous aborderons les instructions de bouclage. Cependant, un exemple littéral pourrait se présenter comme suit.

Lire 100 caractères d'un fichier.
Tant que « compteur » inférieur à 101
< début de boucle >
lit caractère suivant et incrémente le compteur ;
si une touche clavier est pressée **break** ; /* sort de la boucle */

si le caractère lu n'est pas une lettre minuscule **continue** ;
/* retour à la première instruction de la boucle */
sinon transforme minuscule en majuscule ;
< fin de boucle >

* *L'instruction GOTO*

Citons pour être complet l'instruction **goto**, bien que son emploi soit très déconseillé et puisse vous conduire au « plantage » (un saut intempestif risquant de faire perdre au programme son adresse de retour).

goto étiquette ;
...
étiquette : instruction ;

Les instructions composées

Les instructions composées regroupent les instructions conditionnelles, de bouclage et d'aiguillage. Elles ont pour particularité d'être suivies d'une ou plusieurs instructions simples, la syntaxe étant différente selon le cas :

— Une unique instruction suit l'instruction composée.

Le texte est alors écrit en ligne, sans autre ponctuation que le ; terminateur.

if (a>b) i++ ;

Une instruction composée peut n'être suivie que d'une instruction vide.
if (a==b) ; /* si a identique à b, ne rien faire */
while (1) ; /* tant que 1 vrai, ne rien faire : boucle sans fin */

— Une instruction composée entraîne l'exécution de plusieurs instructions.

La suite d'instructions est alors encadrée par une accolade ouvrante et fermante pour constituer un « bloc ». Le juste positionnement de ces accolades est d'une importance extrême et demande une particulière attention lorsque des « blocs » s'imbriquent. Pour faciliter la compréhension et éviter les erreurs, il est utile d'indenter les lignes du programme.

```
if (a>b){
        if (a>c){
                instruction 1;  /*   exécutés seulement si   */
                instruction 2;  /*     a > b  et  a > c      */
                }
        else    {
                instruction 3;  /*   exécutés seulement si   */
                instruction 4;  /*     a > b  et  a <=c      */
                }
else instruction 5; /* exécuté seulement si a <= b */
instruction 6;      /* exécuté dans tous les cas   */
```

* *L'instruction IF-ELSE*

Certainement la plus connue des instructions conditionnelles, elle se présente sous la forme :

if (expression) instruction ; ou (bloc d'instructions)
et facultativement
else instruction ; ou (bloc d'instructions)

ou expression s'entendant au sens large du terme, est évalué. Son résultat VRAI (valeur différente de 0) ou FAUX (égal à 0) détermine les instructions à exécuter.

Lorsque plusieurs **if** sont imbriqués, le **else** éventuellement rencontré peut se révéler ambigu. La règle suivante est alors adoptée : tout **else** s'associe au dernier **if** sans **else** qui le précède, l'utilisation d'accolades pouvant bien entendu modifier ces concordances.

```
if (val) i++;
if (!aff) x = y;
else printf ("COUCOU");
```

```
if (val){ i++;
          if (!aff) x = y;
        }
else printf ("COUCOU");
```

Dans le cas de gauche, la chaîne de caractère sera affichée si **!aff** est FAUX (si aff != 0) ; dans le cas de droite elle sera affichée si **val** est FAUX (== 0).

* *L'instruction WHILE et DO-WHILE*

Cette instruction génère une boucle, exécutée tant que le résultat d'une expression est VRAI.

while (expression) instruction ; ou (bloc d'instructions)

```
while (c=fonc_1())
     /* la fonction fonc_1 retourne une valeur oui est placée
     dans la variable c. Tant que c != 0. la boucle est exécutée */

     {        /* début de boucle  */
     if (condition) continue; /* reiteration immédiate */
     instruction 1;
     if (condition) break;    /* si.... sort de la boucle */
     }        /* fin de boucle */
```

L'instruction **do-while** est utilisée lorsque l'on désire que la condition de rebouclage soit testée en fin de boucle, ce qui signifie que les instructions de la boucle seront exécutées au moins une fois.

```
do {
instruction 1;
instruction 2;
} while (expression); /* si VRAI rebranchement à instruction 1 */
```

* *L'instruction FOR*

Instruction très proche du FOR...TO...STEP du Basic, elle se présente sous la forme :

for (initialisation ; expression ; affectation)
instruction ; ou {bloc d'instruction}

L'initialisation n'est effectuée qu'au départ de la boucle, tandis qu'à chaque passage un test de l'expression est réalisé, entraînant, s'il est VRAI, une nouvelle affectation et un rebouclage.

for (i=0 ; i < 100 ; i++) instruction ; /* exécutée 100 fois */

Les 3 composantes d'un **for** peuvent être multiples ou inexistantes :

for (; x < 50 ; x = y*2, j--) {bloc d'instructions}

Ici il n'y a pas d'initialisation, alors que 2 affectations (séparées par une virgule) sont effectuées.

* *L'instruction SWITCH-CASE-DEFAULT*

Cette instruction d'aiguillage a pour plus proche équivalent Basic le ON...GOSUB, bien qu'en C un branchement ne soit en rien obligatoire : elle pourrait se traduire autrement par SELON...FAIT.

```
switch (expression) {
     case valeur 1: instruction 1;
     case valeur 2: instruction 2;
     ...
     default      : instruction 3;
                    }
```

Le déroulement des instructions débute là ou existe une correspondance entre le résultat de l'expression et la valeur (obligatoirement une constante numérique ou symbolique). L'instruction qui suit **default** (facultatif), sera la seule exécutée si aucun **case** n'est applicable. Les instructions **break** et **return** permettent de sortir du switch.

Il faut bien comprendre que, lorsqu'un **case** est sélectionné, toutes les instructions suivantes sont exécutées jusqu'à la rencontre d'un ordre de sortie du **switch**. Dans l'exemple qui suit, le **break** du **case** "**y**" est obligatoire si l'on ne veut pas voir s'afficher la chaîne de caractères derrière **default**.

```
switch (carac){
    case 'a':printf ("Puisque première lettre de l'alphabet, ");
    case 'e':
    case 'i':
    case 'o':
    case 'u':
    case 'y': printf ("c'est une voyelle.\n");
              break;    /* quitte le switch */
    default : printf ("Ce n'est pas une voyelle minuscule.\n");
            }
```

2.1.7. Directives de compilation

Les directives s'adressent au pré-processeur CP68 et pour cette raison sont précédées du symbole **#**. Elles permettent :

— *une substitution symbolique :* le mot réservé **define** permet de remplacer une chaîne ASCII par celle qui la suit. Les deux chaînes doivent être séparées par un minimum d'un espace. La phase de compilation n'ayant pas encore débutée, tout est permis :

```
#define VRAI  1
#define TANTQUE  while
#define MAX(x,y)  ((x) > (y) ? (x) : (y))
```

— *le chargement d'un autre fichier source :* **include** placé en tête de programme chargera le nom du fichier mis entre guillemets. L'utilisation la plus fréquente consiste à charger un fichier .H, contenant une liste de substitutions symboliques pré-existantes.

```
#include "gemdefs.h"
#include "osbind.h"
```

Nous ne saurions trop vous conseiller d'étudier les fichiers H fournis avec votre compilateur : ils sont souvent riches d'enseignement.

Un autre mot clef du langage C, **typedef**, permet de définir des types synonymes. Ce mot n'est pas interprété par le pré-processeur mais par le compilateur. Il est néanmoins proche de **define,** permettant d'employer un nom défini plutôt qu'un type de variable.

typedef int entier ;
rend synonymes les déclarations : int truc ; et entier truc ;

2.1.8. Les tableaux

Un tableau est une suite d'objets consécutifs et de même type, constituant un ensemble fini (équivalent du DIM en BASIC).

Comme tout objet manipulé par C, un tableau se déclare en précisant son identificateur, le type et le nombre des éléments qui le constituent.

int val[10] ; extern float dec[50] ;

La valeur entre crochets représente *le nombre d'éléments du tableau* et non la valeur du dernier indice comme en BASIC. Les indices de tableaux commençant toujours à zéro, le tableau **val** de l'exemple précédent permet de référencer 10 entiers indicés de 0 à 9.

L'accession aux valeurs des tableaux s'effectue en précisant la valeur d'indice :

val[3] représente le quatrième entier stocké dans le tableau **val.**

Peu importe d'ailleurs le type d'élément du tableau, le compilateur s'occupant de la correspondance entre l'indexation et l'élément spécifié.

```
int x;
float y;

 x = val[1];    /* place dans x les 2ième et 3ième octets par rapport
                   à l'origine du tableau  val  */
 y = dec[1];    /* place dans y du 5ième au 8ième octets relativement
                   à l'origine du tableau  dec  */
```

Une chaîne de caractères est elle aussi stockée dans un tableau, le dernier élément de celui-ci étant la valeur nulle (0) qui permet aux fonctions de détecter la fin de la chaîne. Un tableau de caractères comporte donc toujours un élément de plus que la longueur de la chaîne. Lors des initialisations, le compilateur place automatiquement cette valeur nulle.

char chaîne[7] = " COUCOU " ; /* place les 6 caractères de l'indice 0 à l'indice 5, et chaîne[6] = 0 */

Vous pouvez cependant faire confiance au compilateur pour compter le nombre de caractères et gérer le tableau en conséquence.

char voyelle[] = "AEIOUY" ; /* crée un tableau de 7 éleménts */

Le langage C permet également l'utilisation de tableaux à plusieurs dimensions, que l'on initialise par une liste de valeurs entre accolades.

```
int tab [2][3]= { {0,1,2}, {3,4,5} };

var = tab[1][2];  /* var = 5 */
```

2.1.9. Les pointeurs

Pour bien comprendre l'utilité des pointeurs de variables, il faut se souvenir que nous n'avons vu jusqu'ici que des fonctions qui se transmettaient des arguments par valeur, c'est-à-dire des « photocopies » de variables qui n'influençaient nullement l'original. Cette méthode s'avère fort utile lorsque l'on ne désire pas que la variable d'origine soit modifiée par la fonction appelée. Mais dans le cas contraire ?

L'utilisation des pointeurs va nous permettre d'envoyer à une fonction l'adresse d'une variable afin que la fonction appelée puisse travailler directement sur le contenu de l'adresse de la variable. C'est en fait ce qui se produit systématiquement en Basic : si une variable A est reconnue tout au long du même programme, c'est bien parce que l'interpréteur Basic travaille sur le contenu d'une adresse. Le pointeur qui pose de nombreux problèmes au débutant en langage C, est paradoxalement, ce qu'il connaît le mieux.

Si x est une variable, l'opérateur unaire **&** (à ne pas confondre avec l'opérateur & qui provoque un ET logique entre deux valeurs), permet de connaître l'adresse d'une variable. L'adresse de cette variable est ce que l'on nomme en C un pointeur.

Ainsi, si l'on définit **adr_val** comme un long, écrire :

adr_val = &val ;

signifie que l'on place l'adresse de la variable **val** dans **adr_val,** adresse pointant maintenant sur le contenu de la variable.

Il est possible de connaître l'adresse d'un tableau en demandant l'adresse de son premier indice, ou plus simplement encore en ne considérant que l'identificateur du tableau, ce dernier étant déjà une adresse.

adr_tab = &tab[0] ; est équivalent à adr_tab = tab ;

Il ne suffit pas d'avoir une adresse, encore faut-il pouvoir accéder à son contenu. L'opérateur unaire * est l'opérateur d'indirection spécifiant un contenu d'adresse.

x = * adr _ val ; /* x = contenu de l'adresse adr _ val */

Puisque **adr_val** est l'adresse de **val,** x est maintenant égal à **val.** Partout l'expression ***adr_val** peut remplacer la variable **val** et se comporter exactement comme elle.

*adr_val+ + ; *adr_val = 10 ; if (*adr_val) ... ;

sont toutes des instructions valides.

Un pointeur doit être déclaré pour signaler le type de la variable vers laquelle il pointe.

```
fonc_1 ()  /* titre (ou identificateur) de la fonction */
{
int i =10;

fonc_2 ( &i ); /* envoie à la fonction l'adresse de i */
               /* au retour, i= 11  */
}

fonc_2 (pt)
int *pt;    /* déclaration de pt, adresse pointant sur un entier */
{
*pt++;  /* incrémente le contenu de la variable, soit i */
}
```

2.1.10. Les structures

Une structure est un ensemble de variables, souvent de types différents, regroupées par un même identificateur pour en faciliter le traitement. Une structure simple, employée pour l'appel des fonctions GEM, constituera notre premier exemple.

```
struct grect {
     int g_x;
     int g_y;
     int g_w;
     int g_h;
} ;
```

Le mot réservé **struct** décrit une forme de structure, suivi de sa référence **grect**. Les quatre entiers qui constituent cette structure encadrés par des accolades, sont appelés les **membres**.

La structure **grect** est maintenant définie comme un type, de la même manière que **int** ou **char** sont des types. Déclarer une variable :

struct grect var ;

désignera **var** comme une variable de type **grect**.

Lorsque, dans une expression, on désire s'adresser à un membre de la structure, on utilise la syntaxe suivante :

« variable de la structure » . « membre » Ex : coor_x = var . g_x ;

L'opérateur « point » (.) met en relation la structure et son membre. Il est ainsi possible de s'adresser séparément à chaque membre de la structure considérée pourtant comme un tout. La structure **grect** va pouvoir nous servir à définir quatre éléments nécessaires en GEM pour l'ouverture d'une fenêtre, d'un dialogue, etc... Les quatre membres désigneront respectivement les coordonnées X et Y d'un rectangle, sa largeur et sa hauteur.

Un autre opérateur permet de s'adresser à un membre particulier d'une structure, s'appliquant cette fois à un pointeur de structure. Si l'on déclare un pointeur de structure :

struct grect *p_rec ; /* p_rec est une adresse pointant sur grect */

« pointeur » —> « membre » Ex : coor_x = p_rec->g_x ;

Cette instruction donnera le même résultat que précédemment et affectera à la variable **coor_x** la valeur du membre **g_x**.

L'intérêt des structures est encore plus évident lorsque les membres sont des variables de types différents. L'exemple classique consiste à présenter une structure capable de stocker les caractéristiques d'un individu. Cette structure pourrait comprendre un nom (une chaîne de caractères), un numéro de téléphone (un long), et le sexe (1 ou 2 dans une variable char).

```
struct pers {
  char nom[20];
  long tel;
  char sexe;
  } individu, *pt_ind;
```

Les deux identificateurs qui suivent l'accolade fermante déclarent immédiatement une variable et un pointeur de type **struct pers** et remplace la déclaration : struct pers individu, *pt_ind ;

Là encore, les opérateurs d'adressage "." et "->" permettront de s'adresser directement aux membres de cette structure.

2.2. *Les bibliothèques C*

2.2.1. La bibliothèque standard

Les fonctions de la bibliothèque standard que supporte le compilateur de Digital Research sont détaillées dans l'annexe 1 et nous vous invitons à vous y reporter avant de continuer plus avant la lecture de cet ouvrage : il

est important en effet d'avoir une idée globale de ce que peut apporter cette bibliothèque. Sa connaissance, même succinte, saura vous éviter la résolution, inutile puisque déjà prévue, de problèmes spécifiques.

Le lecteur voudra bien excuser le manque de développement dans l'exposé de ces fonctions uniquement dû au souci d'expliciter plus longuement le GEM, bibliothèque récente et donc bien moins connue. Les ouvrages sur le C (cf bibliographie) apportent déjà de nombreux renseignements sur les fonctions de base du langage et les fournir une fois encore n'aurait été que redondance.

2.2.2. La bibliothèque GEM

VDI et AES forment les deux composantes de la bibliothèque GEM. Si les routines qui les constituent sont effectivement séparées en mémoire (cf chap. 1.2.), si même l'éditeur de liens les dissocie en leur affectant une librairie spécifique, il n'en demeure pas moins vrai que le programmeur peut les considérer comme un ensemble.

Il est encore possible, en se privant de nombreuses facilités, d'écrire une application n'utilisant que le VDI. Mais ce n'est pas vrai pour l'AES qui exige au minimum l'appel de la fonction VDI **v_opnvwk** ouvrant une station de travail virtuelle. Une application ne se fait donc ni en VDI, ni en AES mais bien en GEM. Si ces deux bibliothèques sont également utiles, elles n'en gardent pas moins leurs spécificités.

L'AES est particulièrement orienté vers la gestion des fenêtres, de la souris et des interactions avec l'utilisateur et les autres applications. Pointue et évoluée, une seule fonction AES représente la plupart du temps l'équivalent de plusieurs appels VDI.

On est toujours le « pauvre » de quelqu'un et le VDI, malgré ses grandes capacités, paraît quelquefois terne au regard de la bibliothèque AES, sa voisine. Mais n'est pas toujours dépourvu celui qu'on croit et le VDI pallie bien souvent les manques d'un AES très spécifique. Il n'existe par exemple pas d'équivalent AES pour la sauvegarde ou la restitution de zones d'écran et utiliser les possibilités de l'éditeur alphanumérique interne au VDI peut parfois faire oublier les jolies glissades d'un ascenseur graphique.

Les fonctions VDI

Toutes les fonctions VDI implémentées sur le 520ST sont détaillées dans l'annexe 2, de nombreux renseignements sont également fournis dans le chapitre 3.3. relatif à la ligne "A". Il ne sera expliqué ici que quelques règles relatives à l'appel en C.

Les tableaux des paramètres VDI, explicités en début d'annexe et dont le contenu est spécifié pour chaque fonction, doivent obligatoirement être déclarés comme variables globales en tête de votre fichier source. La capacité des différents tableaux doit au moins être égale aux nombres de paramètres qu'envoie ou que retourne la fonction du programme la plus « gourmande ».

Si vous n'en êtes pas à économiser quelques octets de mémoire, vous pouvez décider de les dimensionner aux valeurs maximales suivantes, permettant sans restriction l'appel de toute fonction VDI :

```
int control [11] ; /* 11 paramètres pour les fonctions vectorisées */
int intin [80] ;   /* longueur de chaîne pour v_gtext */
int ptsin [128] ;  /* valeur maximale de points en entrée */
int intout [45] ;  /* 45 paramètres pour v_opnvwk */
int ptsout [12] ;  /* 12 paramètres pour v_opnvwk */
```

Lorsqu'une fonction fournit ou retourne comme paramètres des coordonnées de points, celles-ci s'évaluent TOUJOURS par rapport au coin supérieur gauche de l'écran représentant le point 0x0. Les coordonnées du coin inférieur droit de l'écran sont dépendantes de la résolution choisie (639x399 si haute résolution, 639x199 si moyenne, 319x199 si basse). Le GEM installé sur l'Atari ne permet pas en effet l'utilisation du système de coordonnées NDC, implémenté sur d'autres machines.

A l'édition de liens, le fichier .0 obtenu après compilation doit être lié avec la librairie **vdibind.**

Les fonctions AES

La nomenclature en annexe 2 explicite les fonctions AES en fournissant pour chacune d'elles le détail des paramètres nécessaires. En C, la concordance entre les identificateurs d'arguments et leurs localisations dans les tableaux internes du système ne sont d'aucune utilité.

Les seules exigences consistent à appeler, avant toute autre, la fonction d'initialisation **appl_init (appl_exit** en fin d'application) et à lier le fichier compilé avec la librairie **aesbind.**

Les pages suivantes donnent des précisions sur les principaux objets que manipule l'AES et explicite l'agencement des fonctions AES pour certaines tâches déterminées.

Note : Les noms symboliques utilisés dans la définition des objets sont inclus dans le fichier « OBDEFS.H ».

Création des ressources :

La première action à envisager lorsque vous programmez en GEM consiste à bâtir votre fichier de ressources. (Vous ne pourrez en effet tester votre programme sans avoir défini, pour le moins, les objets composant votre menu.)

Les ressources définissent de façon structurée tous les objets qu'aura à utiliser une application particulière. Le mot « objet » a un sens très large en GEM et représente aussi bien des chaînes de caractères, des boîtes de dimensions variées contenant généralement d'autres objets, des zones rectangulaires de tailles réduites incluant un texte court (des « boutons »), des icônes, des images, etc...

Ces objets sont classés de façon particulière afin de préciser les relations qu'ils entretiennent entre eux et l'on nomme **arbre** une série d'objets divers, étroitement dépendants les uns des autres. L'objet à la base de l'arbre représente la racine, les « branches » constituant les enfants qui peuvent eux-mêmes être parents d'autres objets.

Exemple : Une application nécessitant une barre de menu et une forme de dialogue nécessitera deux arbres. La racine du premier représentera la barre du menu, parent des titres du menu, eux-mêmes parents des items du menu qui leur correspondent. Le deuxième arbre aura comme racine un objet en forme de boîte, parent d'une chaîne de caractères et de deux boutons ("OK" et "ANNULER").

Chaque objet est défini par une structure d'objet de 24 octets.

STRUCTURE D'OBJET

0	ob_suit	ob_tête
4	ob_taille	ob_type
8	ob_drap	ob_état
12	ob_spec	
16	ob_x	ob_y
20	ob_larg	ob_haut

ob_suit : l'index de l'objet suivant (−1 si racine, pointant vers son frère si pas d'enfant, pointant vers son parent si dernier enfant)

ob_tête : l'index du premier enfant de l'objet (−1 si pas d'enfant)

ob_taille : l'index du dernier enfant de l'objet (−1 si pas d'enfant)

ob_type : un mot représentant le type de l'objet (voir plus loin)
ob_drap : un mot représentant le drapeau d'objet (voir plus loin)
ob_état : un mot représentant l'état de l'objet.
ob_spec : un long mot dont le contenu dépend du type de l'objet
ob_x : coord. X de l'objet relativement à son parent (écran si racine)
ob_y : coord. Y de l'objet relativement à son parent (écran si racine)
ob_larg : largeur de l'objet en pixels
ob_haut : hauteur de l'objet en pixels

Les valeurs de type **(ob_type)** représentant les objets usuellement employés sont les suivantes :

* 20 G_BOX : boîte graphique (ob_spec= couleur et épaisseur de ligne)
* 21 G_TEXT : texte graphique (ob_spec= adresse de struct. TEDINFO)
* 22 G_BOXTEXT : une boîte graphique contenant du texte (ob_spec= adresse d'une structure TEDINFO)
* 25 G_IBOX : une boîte graphique sans couleur de fond (ob_spec= couleur et épaisseur de ligne)
* 26 G_BUTTON : un bouton contenant du texte justifié (ob_spec= adresse de la chaîne texte terminée par 0)
* 27 G_BOXCHAR : un bouton contenant un unique caractère (ob_spec= le caractère + couleur + épaisseur)
* 28 G_STRING : texte graphique (ob_spec= adresse de la chaîne)
* 29 G_FTEXT : texte graphique formaté (ob_spec= adresse TEDINFO)
* 30 G_FBOXTEXT : une boîte contenant du texte formaté (ob_spec= adresse de structure TEDINFO)
* 31 G_TITLE : chaîne utilisée pour les titres de menus (ob_spec= adresse de la chaîne)

La valeur du drapeau **ob_drap** indique à l'AES si le bit correspondant est mis à 1 que :

* 0x0000 NONE : signe particulier = néant
* 0x0001 SELECTABLE : l'utilisateur peut sélectionner l'objet par cliquage → affichage en vidéo inverse
* 0x0002 DEFAULT : l'objet de sortie si touche <RETURN> appuyé : un seul objet par arbre peut être défini ainsi

* 0x0004 EXIT : l'AES retourne le contrôle à l'application si l'utilisateur a cliqué sur l'objet.
* 0x0008 EDITABLE : l'objet est éditable par l'utilisateur : spécifie un champ d'édition
* 0x0010 RBUTTON : un groupe de boutons ne pouvant être sélectionnés en même temps : si l'un est sélectionné, les autres sont désélectionnés par l'AES.
* 0x0020 LASTOB : le dernier objet de l'arbre
* 0x0040 TOUCHEXIT : l'AES retourne le contrôle à l'application si bouton de souris pressé et souris dans l'objet

La valeur de **ob_etat** indique si le bit correspondant est mis que :

* 0x0000 NORMAL : affichage en vidéo normale
* 0x0001 SELECTED : affichage en vidéo inverse
* 0x0002 CROSSED : une croix est tracée sur l'objet (si l'objet = boîte)
* 0x0004 CHECKED : un « check » (√) est tracé près de l'objet
* 0x0008 DISABLED : affichage en intensité réduite
* 0x0010 OUTLINED : l'objet est entouré d'une ligne
* 0x0020 SHADOWED : « ombre » l'objet (dessine un relief)

Nous avons vu que les objets de type G_TEXT, G_BOXTEXT, G_FTEXT et G_FBOXTEXT contenaient dans le long mot **ob_spec** une adresse de structure que GEM nomme TEDINFO. Cette structure de 28 octets définit un champ d'édition, c'est-à-dire un objet géré par l'AES qui laisse l'utilisateur entrer un texte au clavier selon un format et un espace de validité définis dans la structure.

Structure TEDINFO :

0	te_ptext	
4	te_ptmplt	
8	te_pvalid	
12	te_pol	te_resl
16	te_just	te_coul
20	te_res2	te_épais
24	te_txtlg	te_tmplg

te_ptext : l'adresse de la chaîne d'affichage actuel. Si 0 ou "@" comme premier caractère, la chaîne n'est pas affichée.

te_ptmplt : l'adresse de la chaîne de formatage
te_pvalid : adresse de la chaîne de validité. Selon la valeur, autorise :

9 = l'entrée de chiffres
A = l'entrée de lettres majuscules + <espace>
a = lettres majus. et minus. + <espace>
N = lettres majus. + chiffres + <espace>
n = lettres majus. et minus. +chiffres + <espace>
F = caractères valides de nom de fichier + (" ? * : ")
X = tout caractère, converti en majuscule.

te_pol : police système utilisée : 3 = 8x8, 5 = 6x6
te_resl : réservé
te_just : type de justification de texte désirée

0 = justifié à gauche
1 = justifié à droite
2 = centré

te_color : couleur et fond de l'objet
te_res2 : réservé
te_épais : épaisseur de ligne de la boîte.
te_txtlg : longueur de la chaîne pointée par te_ptext
te_tmplg : longueur de la chaîne pointée par te_ptmplt

L'exemple suivant explicite la façon dont travaille l'AES pour gérer un champ d'édition selon les composants de la structure TEDINFO.

— te_ptext pointe sur une chaîne vide
— te_ptmplt pointe sur la chaîne "_____.PRG"
— te_pvalid pointe sur la chaîne "ANNNNNNN"

Quand l'AES entrera dans le champ d'édition, il affichera la chaîne pointée par **te_ptmplt**, placera le curseur et attendra comme premier caractère une lettre majuscule, les six caractères suivants pouvant être indifféremment chiffres ou lettres majuscules (soit sept caractères, un de moins que la longueur de **te_pvalid**).

L'AES gère son curseur ainsi que les touches <ESCAPE> et <BACKSPACE>. Le contrôle est retourné à l'application lorsqu'une condition de sortie est réalisée (cliquage sur une TOUCHEXIT ou <RETURN> si un bouton de sortie est DEFAULT).

Le fichier des ressources est créé à partir du programme RCS.PRG fourni avec certains compilateurs (Digital Research notamment). Cette application permet de construire les différentes structures (d'arbres, d'objets), et fournit un fichier avec l'appendice ".RSC" qui peut être directement chargé par la fonction AES **rsrc_load.**

Si vous avez spécifié un nom pour chaque objet créé (option "Name" du menu), le fichier .H que vous fournira le programme RCS. contiendra les substitutions symboliques (# define). Inclure ce fichier dans votre source (par # include) vous permettra d'utiliser les noms symboliques chaque fois que vous devrez faire référence à un objet.

Une fois chargé le fichier de ressources, en utilisant la fonction **rsrc_gaddr** votre application pourra :

— avec les fonctions de gestion de menu :

* afficher la barre des menus
* modifier les caractéristiques des objets constituant l'arbre du menu (Ex : désactiver un item, placer un « check », etc...)

— avec les fonctions de gestion d'objets :

* tracer un ou plusieurs objets
* modifier l'état d'un objet, etc.

— avec les fonctions de gestion de forme :

* laisser l'AES gérer une forme dialogue.

Une forme dialogue est définie dans un arbre dont la racine est une boîte (type G_BOX) et les enfants, des boutons ou un champ d'édition. Elle permet de solliciter l'utilisateur et d'en recevoir une réponse.

Note : Une forme d'alerte n'est en fait qu'une chaîne « habillant » des objets (une boîte et 3 boutons) appartenant à l'AES. Elle ne fait donc pas partie des ressources.

Lancement de l'application

Lancer une application AES signifie quitter le Bureau, se faire « reconnaître » par le programme maître de l'AES (SCREEN MANAGER) comme nouvelle application courante. C'est le rôle de la fonction **appl_init** qui retourne un identificateur AES dévolu à votre application tant que celle-ci est ouverte.

Après avoir positionné la souris dans sa forme « abeille » par **graf_mouse** spécifiant à l'utilisateur un accès disque, votre application charge par la fonction **rsrc_load** le fichier des ressources, fichier qui devra être placé sur le même disque que votre programme. N'oubliez pas de prévoir un arrêt de l'application et l'affichage d'un message d'erreur au cas où le fichier ne pourrait être chargé. Si tout s'est passé normalement, est retournée l'adresse de stockage de vos ressources.

Un appel à la fonction VDI **v_opnvwork** initialise les paramètres VDI, ouvre la station virtuelle qu'utilisera votre application, et vous retourne un

certain nombre de renseignements. Beaucoup de ceux-ci ne sont utiles que pour assurer une portabilité entre les différentes machines tournant sous GEM, d'autres peuvent être indispensables. Ainsi, connaître la résolution d'écran sélectionnée par l'utilisateur (arg_out[0] et [1]) a souvent une extrême importance pour la suite de votre application. La fonction **vq_extend** peut vous fournir des informations supplémentaires.

L'application réserve par un TRAP 1 fonction $48 (**Malloc**) la place mémoire nécessaire au stockage de l'écran qu'elle aura à gérer. C'est en effet au programmeur qu'il incombe, sur ordre de l'AES, de retracer une zone spécifiée lorsqu'elle a été recouverte par la fenêtre d'un accessoire, une forme dialogue, etc... L'adresse retournée par **Malloc** sera chargée dans le membre fd_addr de la structure MFDB destination (cf chap. 3.3. instr. A007). La MFDB source sera initialisée aux coordonnées d'écran (avec adresse = adresse mémoire écran). Chaque fois qu'il sera nécessaire, le contenu de l'écran sera préservé par l'appel de la fonction VDI **vro_cpyfm**. Il est bien entendu nécessaire de cacher la souris avant la sauvegarde et de la restituer une fois celle-ci effectuée.

Par la fonction **rsrc_gaddr** l'application demande l'adresse de l'arbre définissant le menu. La fonction **menu_bar** affiche la barre des menus dans le haut de l'écran.

L'application crée sa fenêtre. Elle appelle **wind_update** (1) pour éviter toute interférence avec les accessoires. Elle demande par la fonction **wind_get** (identificateur de fenêtre du Bureau = 0, drapeau = 4) la dimension maximale que lui laisse le Bureau, c'est-à-dire la taille de l'écran moins la barre des menus. Comprise dans les limites retournées, la fenêtre est créée par un appel de **wind_create** en spécifiant les composantes de la bordure (case de fermeture, de dimensionnement, etc...). Si la fenêtre a pu être créée, un identificateur est retourné : si (– 1) prévoir une forme d'alerte.

L'application peut maintenant ouvrir la fenêtre en appelant **wind_open** et en précisant les dimensions souhaitées. Elle appelle **wind_update** (0) pour débloquer les accessoires, elle positionne la souris dans sa forme « flèche » puis se met en attente d'événements.

L'attente d'événements :

L'attente des événements est l'axe central de votre application : toute action qu'éxécute celle-ci partira de cette fonction et y reviendra une fois la tâche exécutée pour attendre un nouvel événement.

Vous devez écrire cette fonction comme une boucle sans fin, la seule possibilité de sortie (par l'instruction **break**) devant être le message indiquant

que l'utilisateur désire sortir de l'application, soit en général l'option « Quitter » du menu.

Le chapitre 4.3 explique en détail le fonctionnement du programme maître de l'AES (SCREEN MANAGER). Il suffit de dire ici qu'à chaque fois qu'une application est en attente d'événements, l'AES reprend le contrôle total du système et gère ses interruptions. Chaque interruption est présentée à TOUTE application en attente d'événements (c'est-à-dire à votre application et à chaque accessoire chargé au démarrage du système). Il incombe au programmeur de faire le tri des événements qu'il reçoit.

Une première sélection s'opère d'après l'**evnt** que vous avez choisi : si par exemple vous n'attendez pas d'événement d'horloge, l'AES n'entrera pas dans votre boucle d'attente.

Le second tri s'opère en testant, en même temps que l'événement, le contenu des variables reçues. L'AES spécifie par exemple que l'utilisateur a cliqué dans la case de fermeture d'une fenêtre, il envoie également l'identificateur de la fenêtre à fermer : votre application devra tester ces deux valeurs avant d'entreprendre la moindre action. D'une manière générale, gardez toujours à l'esprit que votre application ne tourne pas seule.

Que se passe-t-il lorsqu'une application et un accessoire attendent le même événement (un appui sur le bouton de souris par exemple) dont il est impossible de connaître le destinataire ? Dans ce cas, c'est le mieux placé sur la liste des programmes en attente (c'est-à-dire l'application ou l'accessoire dont la fenêtre est active) qui gère l'événement. Les différents événements que votre application peut attendre de l'AES sont explicités dans l'annexe 2. Ne sera notifié ici que la manière dont s'effectue le partage des tâches.

L'AES gère seul tout ce qui concerne les bordures de fenêtres, la barre des menus et les menus eux-mêmes lorsqu'ils sont déroulés : dans ces zones, tout déplacement de la souris ou appui du bouton provoque une action spécifique que l'application ne contrôle pas (par exemple le déroulement des menus, la boîte en traits pointillés dessinée lors d'un déplacement de la fenêtre ou de l'ascenseur, etc...). Au relâchement du bouton, l'AES envoie un message particulier dont la réception est assurée par **evnt_mesag** ou **evnt_multi.** L'application, si elle le désire, engage l'action demandée.

L'application a par contre le contrôle du reste de l'écran, c'est-à-dire la zone de travail de sa (ses) fenêtre(s) (fenêtre moins bordures) ainsi que la fenêtre de fond du Bureau (partie grisée sur laquelle viennent s'ouvrir les fenêtres) si celle-ci est encore apparente.

Gestion de dialogue :

Un dialogue s'instaure entre l'application et l'utilisateur chaque fois qu'une action exige une décision de ce dernier.

L'application demande par **rsrc_gaddr** l'adresse de l'arbre de dialogue que le programmeur a défini dans les ressources. Elle positionne les coordonnées d'affichage de la boîte de dialogue (voir « calculatrice ») ou appelle simplement **form_center** qui retourne les coordonnées centrées par rapport à l'écran.

L'application appelle **form_dial** avec le drapeau à 0 réservant ainsi l'espace écran, dessine l'arbre par **objc_draw**, laisse l'AES gérer les interactions entre le dialogue et l'utilisateur par un appel à **form_do**. Lorsqu'un bouton qui a été défini comme EXIT ou TOUCHEXIT (voir drapeau d'objets) est sélectionné par cliquage (ou touche <RETURN> si un bouton est DEFAULT), l'AES retourne l'index d'objet qui a occasionné la sortie.

L'application appelle **form_dial** avec le drapeau à 3 pour libérer l'espace écran, et teste :

— soit uniquement le bouton de sortie
— soit la chaîne contenue dans le champ d'édition s'il existe
— soit l'état des RBUTTON (voir les types d'objets)

Après analyse, l'application désélectionne l'objet de sortie par **obj_change**, entreprend l'action demandée et retourne à la boucle d'attente d'événements où elle recevra un ordre de retraçage.

Retraçage d'écran :

L'AES a envoyé un message de retraçage qui vous est destiné, spéficiant ainsi que la zone de travail dont votre application a la charge doit être remise à jour. Il vous fournit en même temps les coordonnées de la zone à retracer. Il est envisageable cependant que cette zone n'appartienne pas en totalité à votre application. L'intersection de cette zone et de votre fenêtre peut même ne pas vous concerner totalement : si par exemple un accessoire est positionné sur votre fenêtre, retracer l'intégralité de celle-ci recouvrirait tout ou partie d'une zone dont vous n'aviez pas à vous soucier.

Pour éviter ces interférences, l'AES tient à jour une liste de rectangles qui délimite les différents « propriétaires » de l'écran. C'est en appelant cette liste que votre application pourra connaître sa zone effective de retraçage.

L'application appelle d'abord la fonction **wind_update** (1) qui interdit toute interférence, appelle ensuite **wind_get** avec la valeur du drapeau à 11 qui retourne les coordonnées du premier rectangle de la liste. Peut alors commencer une boucle ayant la forme suivante :

1) tant que coord. X et Y retournées différentes de 0
2) s'il existe une intersection entre le rectangle retourné et la zone de travail de l'application alors :
   * structure MFDB source = structure définissant la mémoire avec comme coordonnées la zone d'intersection à retracer.
   * structure MFDB destination = écran (adresse = 0x0L) avec les mêmes coordonnées que la source.
   * appel de **vro_cpyfm** (voir VDI).
3) appel de **wind_get** avec drapeau à 12.
4) retour à 1), condition de boucle.
5) fin de boucle : appel de **wind_update** (0).

Sortie d'application :

Lorsque la condition de sortie de l'application est réalisée, celle-ci sort de sa boucle d'attente d'événement, appelle la fonction de fermeture de fenêtre **wind_close**, la fonction de suppression de fenêtre **wind_delete**. L'application supprime la barre des menus (**menu_bar** avec le drapeau à 0), libère l'espace mémoire réservé pour l'écran par TRAP 1 fonction $49 (Mfree), ferme la station de travail virtuelle (**vclsvwk**), termine l'application par **appl_exit.**

2.3. Exemple de programme

Le programme dont le listing est fourni dans les pages suivantes constitue un exemple concret d'utilisation des fonctions GEM et, dans une moindre mesure, des fonctions de la bibliothèque standard. Son rôle n'est pas seulement didactique puisqu'il vous permettra de disposer d'une calculatrice dans les accessoires du Bureau.

Il vous faut d'abord re-créer le fichier des ressources avec l'utilitaire RCS.PRG. Le listing en C des ressources vous aidera à rebâtir l'unique arbre qu'utilise l'accessoire.

Cet arbre (une forme dialogue) est « père » de 20 « enfants »:

— 19 boutons de type G_BUTTON (les dix chiffres de 0 à 9, les quatre opérateurs élémentaires, le point décimal, le signe "=", un bouton "CE", un bouton "CA" et le bouton "FIN" qui stoppe la calculatrice).
— 1 champ d'édition de type G_FBOXTEXT, l'écran de la calcultratice.

Tous les boutons sont des TOUCHEXIT, SELECTABLE, le bouton "=" étant également DEFAULT. La chaîne de validité (te_pvalid) doit contenir exclusivement des "X".

Vous donnerez à chaque objet de l'arbre (option "Name" de RCS) le nom défini dans le fichier "CALCUL.H", mais il faut également que les 10 index définissant les boutons de chiffres soient dans un ordre croissant et régulier. Selon la valeur d'index du bouton "0" (dans le programme, elle est à 1), il vous faudra modifier dans la fonction **gere_cal** la ligne **default : val = sortie + 47 ;** qui ajoute une valeur à l'index d'objet pour retrouver les codes ASCII des chiffres 0 à 9.

Une fois le fichier "CALCUL. RSC" créé, vous pourrez en demander le listing C pour vérifier qu'il correspond bien au modèle. Les différences entre les tailles des objets peuvent toutefois être comprises dans une certaine fourchette.

Après saisie et compilation, le liage du fichier "calcul.0" que vous aurez obtenu s'effectuera avec la commande suivante :

link68 [tem[a ou b :], u, s,] calcul.68k= accstart, osbind, calcul.0, aesbind, vdibind, gemlib, libf

Le fichier "calcul.68k" sera ensuite rendu relogeable avec "relmod".

Le programme obtenu doit être placé sur la disquette système avec, comme appendice de nom de fichier, les trois lettres .ACC. Le fichier "Calcul.rsc", ressources de l'accessoire, devra également être chargé sur la disquette système.

MODE D'EMPLOI : En cliquant sur l'item du menu "Calcul", vous ouvrez l'accessoire, le déplacement de la fenêtre est alors valide. En cliquant sur la case "pleine fenêtre", vous branchez la calculatrice : le message "ON" s'affiche dans le bouton interrupteur et le curseur est affiché. Clavier et souris autorise l'entrée de caractères. Le bouton "CE" efface le dernier chiffre entré, le bouton "CA" efface l'intégralité de la chaîne. Tout caractère invalide sera refusé par l'application. Pour éteindre la calculatrice, il faut cliquer dans la touche "ON" et cliquer sur la case de fermeture pour refermer l'accessoire.

Cette calculatrice n'utilise que la simple précision (flottant), le type **double** n'étant pas implémenté sur le compilateur de Digital Research. Elle n'est d'autre part pas prévue pour fonctionner en basse et moyenne résolution : la gestion des modifications des tailles des ressources aurait en effet considérablement alourdi le programme. Mais il n'est pas interdit, bien au contraire, de l'améliorer.

FICHIER DES DEFINITIONS

```
#define CALCUL 0     /* TREE */
#define ZERO 1       /* OBJECT in TREE #0 */
#define UN 2         /* OBJECT in TREE #0 */
#define DEUX 3       /* OBJECT in TREE #0 */
#define TROIS 4      /* OBJECT in TREE #0 */
#define QUATRE 5     /* OBJECT in TREE #0 */
#define CINQ 6       /* OBJECT in TREE #0 */
#define SIX 7        /* OBJECT in TREE #0 */
#define SEPT 8       /* OBJECT in TREE #0 */
#define HUIT 9       /* OBJECT in TREE #0 */
#define NEUF 10      /* OBJECT in TREE #0 */
#define POINT 11     /* OBJECT in TREE #0 */
#define EGAL 12      /* OBJECT in TREE #0 */
#define PLUS 13      /* OBJECT in TREE #0 */
#define MOINS 14     /* OBJECT in TREE #0 */
#define MUL 15       /* OBJECT in TREE #0 */
#define DIV 16       /* OBJECT in TREE #0 */
#define ECRAN 19     /* OBJECT in TREE #0 */
#define FIN 20       /* OBJECT in TREE #0 */
#define CA 17        /* OBJECT in TREE #0 */
#define CE 18        /* OBJECT in TREE #0 */
```

FICHIER DES RESSOURCES

```
#define TOOBJ 0
#define FREEBB 0
#define FREEIMG 0
#define FREESTR 22

BYTE *rs_strings[] = {
"0",
"1",
"2",
"3",
"4",
"5",
"6",
"7",
"8",
"9",
".",
"=",
"+",
"-",
"*",
"/",
" CA ",
" CE ",
"",
"  _____________   ",
"XXXXXXXXXXXXX",
"OFF"};



LONG rs_frstr[] = {
0};

BITBLK rs_bitblk[] = {
0};

LONG rs_frimg[] = {
0};
```

```
ICONBLK rs_iconblk[] = {
0};

TEDINFO rs_tedinfo[] = {
18L, 19L, 20L, 3, 6, 0, 0x1180, 0x0, -1, 1,20};

OBJECT rs_object[] = {
-1, 1, 20, G_BOX, NONE, NORMAL, 0x21100L, 0,0, 24,15,
2, -1, -1, G_BUTTON, 0x5, NORMAL, 0x0L, 3,13, 3,1,
3, -1, -1, G_BUTTON, 0x5, NORMAL, 0x1L, 3,11, 3,1,
4, -1, -1, G_BUTTON, 0x5, NORMAL, 0x2L, 8,11, 3,1,
5, -1, -1, G_BUTTON, 0x5, NORMAL, 0x3L, 13,11, 3,1,
6, -1, -1, G_BUTTON, 0x5, NORMAL, 0x4L, 3,9, 3,1,
7, -1, -1, G_BUTTON, 0x5, NORMAL, 0x5L, 8,9, 3,1,
8, -1, -1, G_BUTTON, 0x5, NORMAL, 0x6L, 13,9, 3,1,
9, -1, -1, G_BUTTON, 0x5, NORMAL, 0x7L, 3,7, 3,1,
10, -1, -1, G_BUTTON, 0x5, NORMAL, 0x8L, 8,7, 3,1,
11, -1, -1, G_BUTTON, 0x5, NORMAL, 0x9L, 13,7, 3,1,
12, -1, -1, G_BUTTON, 0x5, NORMAL, 0xAL, 8,13, 3,1,
13, -1, -1, G_BUTTON, 0x7, NORMAL, 0xBL, 13,13, 3,1,
14, -1, -1, G_BUTTON, 0x5, NORMAL, 0xCL, 18,13, 3,1,
15, -1, -1, G_BUTTON, 0x5, NORMAL, 0xDL, 18,11, 3,1,
16, -1, -1, G_BUTTON, 0x5, NORMAL, 0xEL, 18,9, 3,1,
17, -1, -1, G_BUTTON, 0x5, NORMAL, 0xFL, 18,7, 3,1,
18, -1, -1, G_BUTTON, 0x5, NORMAL, 0x10L, 3,5, 6,1,
19, -1, -1, G_BUTTON, 0x5, NORMAL, 0x11L, 15,5, 6,1,
20, -1, -1, G_FBOXTEXT, EDITABLE, NORMAL, 0x0L, 3,1, 18,1,
0, -1, -1, G_BUTTON, 0x25, NORMAL, 0x15L, 3,3, 8,1};

LONG rs_trindex[] = {
0L};

struct foobar {
        WORD      dummy;
        WORD      *image;
        } rs_imdope[] = {
0};

#define NUM_STRINGS 22
#define NUM_FRSTR 0
#define NUM_IMAGES 0
#define NUM_BB 0
#define NUM_FRIMG 0
#define NUM_IB 0
#define NUM_TI 1
#define NUM_OBS 21
#define NUM_TREE 1

BYTE pname[] = "CALCUL.RSC";


/*****************************************/
/*              CALCULATRICE             */
/*****************************************/

/*      Fichiers inclus         */

#include <stdio.h>
#include "gemdefs.h"
#include "calcul.h"
#include "obdefs.h"


/*       Definitions            */

# define PASFEN (-1)      /* fenetre non ouverte */


/*        Variables             */
```

```
extern char  __atab[];   /* pour eviter erreur a l'edition de liens */
extern int   gl_apid;    /* identificateur AES */

            /* Tableaux des parametres VDI */
int     contrl[11];      /* pour stockage des parametres fonctions  */
int     intin[80];       /* pour parametres entiers en entree */
int     ptsin[128];      /* pour coordonnees entieres de pts en entree */
int     intout[45];      /* pour parametres entiers en sortie */
int     ptsout[12];      /* pour coordonnees entieres.de pts en sortie */

long arbre;              /* adresse de l'arbre RESSOURCES */
int  menu_id;            /* identif. de l'accessoire */
int  id_vdi;             /* identif. VDI de l'accessoire */
int  id_fen;             /* identif. de la fenetre de l'accessoire */
int  id_sta;             /* identif. de la station */
int  mess_tp[8];         /* tampon ou sont stockes les messages */
int cpt;                 /* compteur de caracteres de calculatrice */

int     ht_fen = 260;    /* hauteur de fenetre   */
int     lg_fen = 188;    /* largeur de fenetre   */
int     x_fen = 220;     /* coord. X de fenetre */
int     y_fen = 80;      /* coord. Y de fenetre */

char    *fen_titre = " CALCULATRICE "; /* titre de la fenetre */
char    chai_ec [13];                  /* chaine entree a l'ecran */
char    aff_on [4]= "ON ";             /* bouton MARCHE */
char    aff_off [4]= "OFF";            /* bouton ARRET  */

/*********************************************************************/

/* Gestion de la calculatrice */

gere_cal()
{
int j;
int sortie;                            /* objet occasionnant la sortie */
long ted_adr;
char val;

  ted_adr=((long) *((long *)(arbre+12+24*ECRAN)));
  (* ((long *)(ted_adr))=chai_ec);
  ((int) *((int *)(ted_adr+24))=13);
            /* l'adresse de "chai_ec" est place dans la struct. TEDINFO */
                  /* le champ d'edition pointe maintenant sur "chai_ec" */


(*(long *)(arbre+12+24*FIN)) = aff_on;          /* modifie bouton OFF */
trace_obj (FIN);
cpt=(-1);
while (1)                     /* sortie de boucle si cliquage bouton ON */
{
sortie=form_do (arbre,ECRAN);
if (sortie== FIN) (*(long *)(arbre+12+24*FIN)) = aff_off;       /* OFF */
desel_obj (sortie);                /* objet de sortie en video normale */
if (sortie== FIN) {
   for(j = 0; j < 14; j++) chai_ec[j]=0; break;  }     /* vide l'ecran */
switch(sortie){
      case EGAL: if (travail()) break;              /* fait le calcul */
      case CA: for(j=0;j<14;j++) chai_ec[j]=0;     /* efface la chaine */
         cpt=(-1); break;                        /* reinitialise le compteur */
      case CE: if (cpt >=0) chai_ec[cpt--]=0; break;
      default: if (cpt >11) break;
      cpt++;
                  switch (sortie) {
                  case MUL: val='*';  break;
                  case DIV: val='/';  break;
                  case PLUS: val='+'; break;
                  case MOINS: val='-';break;
                  case POINT: val='.';break;
```

```
                default: val=sortie+47; break;/*  pour chiffres          */
                                }              /*  fin SWITCH secondaire   */
                chai_ec[cpt]=val;              /* ajoute chiffre a l'ecran */
                }                              /*  fin du SWITCH principal */
  trace_obj (ECRAN);                            /* retrace nouvel ecran    */
  }                                             /* boucle WHILE            */
  trace_obj (FIN);                              /* place chaine OFF        */
  trace_obj (ECRAN);                            /* affiche un ecran vide   */
}

/* Definit les operandes, l'operateur. Au retour, resultat dans chai_ec  */

travail()
{
char op_1[14];                                  /* chaine pour premier operande */
char op_2[14];                                  /* chaine pour deuxieme operande */
char op_eur;                                                    /* operateur */
char c;
int i,j,res;
int lg_1,lg_2;                                  /* longueurs des deux operandes  */
int lg_max,prec_max;
int dec_1=0;                                    /*   nombre de decimaux des      */
int dec_2=0;                                    /*      deux operandes           */

  for (i=0; (c=chai_ec[i]) && (est_nb (c, &dec_1)) && (i<14); i++)
                                op_1[i]=chai_ec[i];
  op_1[i]=0;                                /* termine la chaine par valeur nulle */
  lg_1=i-1;                                          /* longueur du 1er operande */
  if ((i) && (est_opp(chai_ec[i]))) op_eur=chai_ec[i++];
  else return (0);        /* erreur: pas d'operateur ou pas de premier op. */
  for(j=0; (c=chai_ec[i]) && (est_nb (c, &dec_2)) && (i<14);i++,j++)
                                op_2[j]=chai_ec[i];
  if (!j) return (0);                        /* erreur:pas de deuxieme operande */
  op_2[j]=0;                                 /* termine chaine par valeur nulle */
  lg_2=j-1;
  lg_max=((lg_1>lg_2) ? lg_1 :lg_2);                /* longueur maxi d'operande */
  prec_max=((dec_1>dec_2) ? dec_1 :dec_2);          /* definit precision max. */
  res=operation (prec_max-1, op_1, op_2, &op_eur);    /* fait l'operation */
  sup_zero ();                          /* supprime les zeros non significatifs */
  return (res);                              /* retourne 1, validant le resultat */
}


/* Calcule le resultat de l'operation     */

operation (precis,pre_op,deu_op,op_r)
int precis;                                                /* precision demandee */
char pre_op[];                                             /* premier operande   */
char deu_op[];                                             /* deuxieme operande  */
char *op_r;                                                /* operateur          */
{
float res_1,res_2,resul;
float atof();                  /* declarations des fonctions de bibliotheque: */
float ftoa();             /* obligatoires puique ne retournant pas un entier */

  res_1=atof (pre_op);                        /* convertit une chaine en flottant */
  res_2=atof (deu_op);
        switch(*op_r){
                case'*':resul=res_1 * res_2;
                        precis=((precis>2)? 3 :(precis+1)); break;
                case'/':if (res_2==0) return(0);
                        resul=res_1 / res_2;
                        precis=3;   break;
                case'+':resul=res_1 + res_2;  break;
                case'-':resul=res_1 - res_2;  break;
                }                                          /* fin du SWITCH */
  ftoa (resul,chai_ec,precis);                /* convertit un flottant en chaine */
  return (1);                                                /* pas d'erreur */
}
```

```
/* Teste si chiffre ou ".", compte nombre de decimaux  */

est_nb (carac,decimal)
char carac;
int *decimal;                                   /* compteur de decimaux */
{
int rep=0;

  if((carac>='0') && (carac<='9')) rep=1;       /* "carac" est un chiffre */
  if (*decimal) (*decimal)++;              /* si chiffre decimal, compteur+1 */
  if ((!(*decimal)) && (carac=='.')) {
        (*decimal)=1;                /* rencontre du ".": debut du compteur */
        rep=1;                                  /* caractere "." valide */
        }
  return(rep);                     /* si 0, erreur; si 1, caractere valide */
}


/* Teste si operateur    */

est_opp(carac)
char carac;
{
         switch(carac){
                case '+':
                case '-':
                case '*':
                case '/': return(1);                /* c'est un operateur */
                default : return(0);                /* ce n'en est pas un */
                        }
}

/* Supprime les zeros non significatifs */

sup_zero()
{
int i,j;
char c;

  for( i = 0;(c = chai_ec[i]); i++) /* tant que fin de chaine non trouvee */
                if (c=='.')  j=i;          /* j= position du point decimal */

  cpt = i-1;                                        /* compteur de chai_ec  */
  if (!j) return(0);                        /* il n'y a pas de point decimal  */
  for(i-=1; i>=j; i--) {
         if ((chai_ec[i]=='0')||(chai_ec[i]=='.'))
                 {
                 chai_ec[i] = 0;/* supprime les zero depuis fin de chaine */
                 cpt--;
                 }
        else break;                /* si chiffre significatif trouve: STOP */
                        }                                /* fin de boucle FOR  */
}


/* Deselectionne et trace l'objet "quel"  */

desel_obj (quel)
int quel;
{
  int x_clip,y_clip,lg_clip,ht_clip;

  wind_get (id_fen, 4, &x_clip, &y_clip, &lg_clip, &ht_clip);
  objc_change (arbre, quel, 0, x_clip, y_clip, lg_clip, ht_clip, 0, 1);
}


/*  Redessine l'objet "quel" sans modifier son etat */
```

```
retrace_cbj (quel)
int cual;
{
  int x_clip, y_clip, lg_clip, ht_clip;

  wind_get (id_fen, 4, &x_clip, &y_clip, &lg_clip, &ht_clip);
  objc_draw (arbre, quel, 0, x_clip, y_clip, lg_clip, ht_clip);
}


/* Positionne la boite de dialogue aux coordonnees de la fenetre */

positionne ()
{
int x_dial, y_dial, lg_dial, ht_dial;

  wind_get (id_fen, 4, &x_dial, &y_dial, &lg_dial, &ht_dial);
                                /* demande les coordonnees de la fenetre */
  *((int *) (arbre+16))) = x_dial;
          /* modifie dans la structure d'objet la coord. X de la boite */
  *((int *) (arbre+18))) = y_dial;
          /* modifie dans la structure d'objet la coord. Y de la boite */
}


/* Gestion des messages recus */

gere_ouv()
{
int rep;   /* Cette boucle n'est pas sans fin pour le systeme: les autres */
      /* accessoires et l'application courante sont egalement interroges */
  while (1)
  {
        evnt_mesag (mess_tp);                    /* attente d'un message */
        wind_update(1);                     /* interdit les interferences */
        switch (mess_tp [0])
        {
        case WM_FULLED:     if(mess_tp[3]==id_fen) gere_cal();
                  break;
        case WM_REDRAW:     if (mess_tp[3]==id_fen)
            retrace (mess_tp[4], mess_tp[5], mess_tp[6], mess_tp[7]);
                  break;
        case WM_MOVED: if(mess_tp[3]==id_fen) {
       wind_set(id_fen, 5, mess_tp [4], mess_tp[5],mess_tp[6], mess_tp[7]);
       positionne ();   }            /* place "arbre" aux coord. fenetre */
                  break;
        case AC_OPEN:
       if (mess_tp[4] == menu_id) {
         if (id_fen == PASFEN) {          /* si fenetre non deja ouverte */
              ouv_sta ();                                /* ouvre station */
              if (rep = ouv_fen()) fin_app(rep);
                                  }
         else wind_set (id_fen, WF_TOP,0,0,0,0);    /* sinon rend active */
         }   break;
        case WM_CLOSED: if (mess_tp[3]==id_fen) fin_app (2);
                        break;
       case AC_CLOSE:  if ((mess_tp[3]==menu_id) && (id_fen != PASFEN))
                              fin_app (1);      /* ferme l'accessoire */
                        break;
       case WM_NEWTOP:
       case WM_TOPPED: if (mess_tp[3] == id_fen) {
               wind_set (id_fen, WF_TOP, 0, 0, 0, 0);}
               break;

       }   wind_update(0);                          /* fin de boucle SWITCH */
  }                                                        /* boucle WHILE */
}

/* Retracage de la fenetre        */
```

```
retrace (x, y, lg, ht)
int x,y,lg,ht;                          /* coordonnees du rectangle a retracer */
{
  GRECT source, dest, boite;      /* structure de rectangle (cf DBDEFS.H) */

    source.g_x = x;  source.g_y = y;
    source.g_w = lg; source.g_h = ht;
    wind_get (id_fen, WF_FIRSTXYWH, &boite.g_x, &boite.g_y, &boite.g_w,
                &boite.g_h);   /* demande premier rectangle de la liste */
    while (boite.g_w && boite.g_h) {
       dest.g_x = boite.g_x;  dest.g_y = boite.g_y;
       dest.g_w = boite.g_w;  dest.g_h = boite.g_h;
       if (rc_intersect (&source, &dest))   /* si intersection, retracer */
           objc_draw (arbre, 0, 2, dest.g_x, dest.g_y,
                        dest.g_w, dest.g_h);
       wind_get (id_fen, WF_NEXTXYWH, &boite.g_x, &boite.g_y, &boite.g_w,
                  &boite.g_h);  /* demande rectangle suivant de la liste */
                                  }              /* fin de boucle WHILE */
}

/*  Fin d'accessoire: ferme fenetre et station selon "type_ret" */

fin_app(type_ret)
int      type_ret;
{
  switch(type_ret)
         {
         case 2:
          wind_close (id_fen);                        /* ferme la fenetre */
          wind_delete(id_fen);                     /* supprime la fenetre */
         case 1:
          v_clsvwk(id_vdi);          /* ferme la station virt. de travail */
          id_fen = PASFEN;
           break;
        }                                                /* fin du SWITCH */
}                                        /* retourne attendre un evenement */

/* Creation et ouverture de fenetre */

ouv_fen ()
{
    id_fen = wind_create (0x000f, x_fen, y_fen, lg_fen, ht_fen);
          /* cree une fenetre avec barre de deplacement, case de sortie, */
                                          /* et case de "pleine fenetre" */
    if (id_fen == (-1)) {                   /* si creation impossible ...*/
form_alert (1,"[3][GEM n'a plus aucune|fenetre en reserve![Desole]");
                    return (1); }
    wind_set(id_fen,2,(long)fen_titre,0,0);     /* place titre de fenetre */
    wind_open (id_fen, x_fen, y_fen, lg_fen, ht_fen);
    graf_mouse(0,0x0L);
    positionne ();          /* place l'arbre a la position de la fenetre */
    return (0);                                          /* tout va bien */
}

/*  PROGRAMME PRINCIPAL: initialisation de l'accessoire */

main()
{ int v,w,x,y;                                  /* variables inutilisees */
  appl_init();
  id_sta = graf_handle (&v, &w, &x, &y);
  if(!rsrc_load("CALCUL.RSC"))   /* si fichier de ressources introuvable */
         {
                    graf_mouse(0,0x0L);
  form_alert(1,"[3][Erreur Fatale !|CALCUL.RSC|Fichier inconnu][Arret]");
  appl_exit();                                /* fermeture de l'accessoire */
         }
  menu_id = menu_register(gl_apid, "  Calcul ");
  id_fen = PASFEN;
  rsrc_gaddr (0, CALCUL, &arbre);        /* demande l'adresse de l'arbre */
```

```
  evnt_mesag ();              /* attente des evenements, n'en revient jamais */
}

/* Ouvre station de travail virtuelle */

open_sta ()
{ int arg_in[11], arg_out[57], i;
  id_vdi = id_sta;
  for(i=0;i<10;i++) arg_in[i]=i;
  v_opnvwk(arg_in, &id_vdi, arg_out);
}
```

2.4. Erreurs de programmation

Le défaut majeur des langages compilés tient sans doute à l'absence d'analyse syntaxique et lexicale lors de l'édition du programme. Ces erreurs ne sont détectées qu'à la compilation, parfois lors de l'édition de liens seulement, à moins que l'on n'utilise un programme de « lint » qui, d'ailleurs, ne détectera que les erreurs de syntaxe les plus grossières. Or la compilation est une phase longue surtout pour des programmes importants, de plus l'accumulation d'erreurs (ou certaines erreurs spécifiques) provoquent fréquemment un « plantage » du compilateur, les erreurs ultérieures n'étant plus traitées.

Pour toutes ces raisons il est particulièrement conseillé d'éviter les erreurs lorsque l'on programme en C ! Aussi analyserons-nous ici quelques types d'erreurs parmi les plus fréquentes ou les plus « vicieuses », le lecteur pourra par ailleurs se reporter en 3.1.4. pour un exposé des règles générales de programmation. Les pages qui précèdent ayant apporté les éléments de base de la programmation en C, nous ne saurions que vous conseiller de vous reporter à d'autres ouvrages pour approfondir vos connaissances tout en vous recommandant de toujours garder à l'esprit les spécificités de chaque compilateur C. A titre d'exemple, sur le 520ST seul l'emploi de 3 variables « registre » est autorisé (correspondant à D7, D6 puis D5). Notons d'autre part que le nom de variable ou de fonction C n'est identifié que par ses *SEPT* premiers caractères (ainsi « fichierx'' et ''fichiery'' sont équivalents pour le compilateur...).

2.4.1. Erreurs de syntaxe

Les erreurs de syntaxe résultent bien souvent d'oublis :

— oubli du symbole ";" après un appel de fonction ou une initialisation de variable.
— oubli d'une accolade de fermeture d'une fonction ou d'une instruction.

— ":" au lieu de ";", souvent une faute de frappe due à la contiguité de ces deux touches et délicate à détecter.
— oubli de parenthèse, surtout dans une expression complexe.

Ces erreurs, les plus habituelles, sont repérées à la compilation mais provoquent souvent un dérapage de celle-ci, l'oubli d'une accolade conduisant par exemple le compilateur à inclure les instructions suivantes dans la fonction précédente, d'où une suite de messages d'erreurs... Pour les éviter, l'attention est la meilleure arme mais certains « trucs » sont utiles :

— demander à l'éditeur de rechercher toutes les occurrences de ":" et vérifier qu'il s'agit d'expressions conditionnelles ou de « case ».
— demander à l'éditeur de rechercher toutes les occurrences de ")<CR>", lesquelles ne doivent survenir qu'après des noms de procédures ou après une instruction du type « if », « for », « while » ou « switch ».
— indiquer en commentaire à côté des accolades fermées la nature de la boucle ou de la suite d'instructions ainsi clôturée.

Une erreur de syntaxe particulièrement habituelle chez les programmeurs ayant débuté en Basic est l'utilisation de : if(<expression1> = <expression2>) au lieu de : if(<expression1> = = <expression2>). Cette erreur est redoutable parce qu'en fait non contraire à la syntaxe donc non détectée à la compilation. Or "<expression1> = <expression2>" étant vrai tant que <expression2> est vrai, puisqu'il s'agit d'une affectation, les conséquences de cette erreur sont souvent funestes.

Attention également aux opérateurs accolés au signe égal, certains compilateurs acceptant par exemple "=&" au lieu de "&=" ou le signalant par un simple « warning ». Or si, dans votre esprit, le "&" suivant l'égalité indiquait une adresse, il sera compilé comme un ET logique... Laissez donc toujours un espace après le signe d'égalité, cela vous évitera quelques déboires !

2.4.2. Méconnaissances

Même le programmeur en C le plus chevronné confond parfois un pointeur et un pointeur de pointeurs, ou le contenu et le contenant. Une règle de bon sens voudrait que l'on ne programme qu'avec les outils dont on a acquis une parfaite maîtrise... malheureusement, ces outils sont parfois insuffisants et les progrès naissent des expériences, même cruelles. Aussi les méconnaissances sont-elles à l'origine de nombre d'erreurs.

Pour les éviter, un seul remède : étudier ! Cent fois sur le métier... N'ayez pas de honte à consulter les ouvrages de base sur le C (cf. bibliographie en fin d'ouvrage), il n'est pas rare de voir des développeurs pourtant aguerris

vérifier la syntaxe de telle ou telle instruction. Au-delà des instructions C, au demeurant peu nombreuses, ce sont surtout les pointeurs et les structures qui provoquent des erreurs. Sur les pointeurs, les pages précédentes offrent un aperçu mais nous ne saurions trop vous conseiller la prudence quant à leur utilisation si vous débutez. Mieux vaut parfois créer un tableau de caractères que risquer d'utiliser un pointeur de chaîne à mauvais escient !

Les structures sont parfois indispensables (ainsi la structure FILE pour la bibliothèque standard C ou les structures GRECT et OBJECT pour les bibliothèques GEM). Pour mieux les connaître, il est INDISPENSABLE de lister et d'étudier les fichiers de définitions fournis avec le compilateur (en particulier "stdio.h" et "obdefs.h"). Relevons à ce propos que la structure « FILE » utilisée en CP/M 68k n'a que peu à voir avec la structure « FILE » d'UNIX telle qu'elle est décrite dans l'ouvrage des « pères » du C !

La structure FILE citée plus haut incite à une remarque d'un ordre voisin : il ne faut pas confondre un fichier (ouvert par « open », fermé par « close », lu et écrit par « read » et « write ») avec un canal (ouvert par « fopen », fermé par « fclose », lu et écrit par « fread » et « fwrite »). L'un comme l'autre permettent de traiter *un périphérique* ou un fichier disquette mais, alors qu'un fichier est repéré par un numéro (id_fich), un canal est repéré par un pointeur de structure (FILE, voir fopen, Annexe 1). D'autre part, à un canal est associé un tampon, ce qui autorise un accès séquentiel contrôlé par le pointeur de position, alors qu'un fichier est à accès direct (même si « lseek » autorise un accès séquentiel indirect).

2.4.3. Les pièges

Il n'y a pas de pièges en C... tout juste des méconnaissances, des incompréhensions, des zones d'ombre et des erreurs de programmation ! Le C a toutefois des particularités dont l'importance est parfois mésestimée. De plus les erreurs signalées dans ce paragraphe ne sont habituellement pas détectées à la compilation et difficilement repérables à l'exécution, sinon par le fait qu'elles engendrent les résultats les plus inattendus.

L'une des cause d'erreurs « vicieuses » les plus classiques tient à la confusion entre types de variables et spécialement entre le type « char » et le type « int » ainsi qu'entre les types signés et non signés. Ce type d'erreur fera sourire le programmeur confirmé mais certains cas de confusion sont assez traîtres. Ainsi :

```
int mldiv(a,b,c)
int a,b,c;
{
    return ( (a*b)/c );
}
```

se traduira par une multiplication des entiers a et b avec un résultat *ENTIER*, lequel résultat entier sera ensuite étendu à un entier long puis divisé par c. La fonction correcte est :

```
int muldiv(a,b,c)
int a,b,c;
{
    return ( ( (long) a*b)/c );
}
```

La confusion des types atteint des sommets lorsque l'on utilise une variable définie comme de type « char » pour servir d'indice de tableau, par exemple pour convertir un caractère à partir d'une table de conversion. Car si, par hasard, la variable indice est supérieure à 127 (donc un caractère négatif) on obtiendra *un indice négatif*, car un indice est TOUJOURS ENTIER et le caractère négatif sera étendu à un entier négatif. Inutile de décrire le tableau !

Les erreurs d'incrémentation et de décrémentation sont aussi courantes. Rappelons que l'incrémentation d'un pointeur se traduit par un ajout de 1 si le pointeur pointe des caractères, de 2 s'il pointe des entiers, de 4 s'il pointe des longs mots. Cette règle simple et parfaitement logique est parfois délicate à appliquer. Ainsi :

char ** fichlist ;

définit un pointeur pointant sur des pointeurs de caractères. Donc :

fichlist += 4;

provoquera une incrémentation de *16* de fichlist.

Quelques règles de bon sens doivent aussi rester présentes à l'esprit. Ainsi le contraire de ">" n'est pas "<" mais, bien sûr, "<="... Enfin le processus de compilation (voir chapitre 4) et ses particularités doivent être bien appréhendés. Il faut par exemple savoir que le compilateur Digital Research convertit l'opérateur de décalage à droite ">>" par l'instruction 68000 « asr », ce qui conduit pour un nombre non déclaré comme « unsigned », et de bit fort à 1, à un remplissage des bits forts par des « 1 ». Il est également préférable d'éviter l'emploi de noms de variables ou de procédures précédés d'un trait de soulignement "_", certaines variables système utilisant ce préfixe. D'une façon générale, mieux vaut utiliser des noms de variables ou de fonctions français, c'est encore ainsi que l'on risque le moins de définir un nom déjà défini par le système !

2.4.4. Les confusions de bibliothèques

Le souci d'être compris, et peut-être un certain manque d'imagination, a amené les concepteurs des bibliothèques C à utiliser des noms de fonctions très proches, source de confusions multiples pour le débutant qui en vient souvent à reléguer au placard toute une bibliothèque de peur des erreurs de manipulation ou simplement parce qu'il a l'impression qu'elle fait double emploi avec une autre...

Seule une dyslexie accentuée peut provoquer la confusion entre l'instruction C « break » et la fonction C « brk() », mais il peut arriver de confondre la fonction standard « malloc ( ) » et la fonction gemdos « Malloc ( ) » ou la fonction standard « fopen » et la fonction gemdos « Fopen ». Or cela n'est pas sans conséquences ! Ainsi « malloc » utilise un nombre ENTIER pour réserver une zone mémoire alors que « Malloc » utilise un ENTIER LONG, permettant ainsi de réserver une zone de plus de 64Ko. Quant à l'instruction « fopen », elle ouvre un CANAL et retourne un *pointeur de structure FILE*, alors que « FOPEN » ouvre un FICHIER et retourne un identificateur de fichier. On pourrait penser qu'en conséquence « Fopen » est identique à « open »... et effectivement les paramètres d'appel sont identiques mais « Fopen » retourne un identificateur LONG alors que « open » retourne un identificateur ENTIER. Cet identificateur étant utilisé pour l'accès et la fermeture du fichier, ces fonctions ne sont pas interchangeables !

Il existe des cas de double emploi, ainsi la fonction C « sbrk » est identique à Pterm0 (Trap 1, fonction 0), elle-même identique à Pterm(0) (Trap 1, fonction S4C), encore identique à Trap 2 avec D0 = 0 ! Ces cas sont toutefois l'exception et, bien souvent, les fonctions système (analysées en Annexe 4) apportent des améliorations ou des facilités d'emploi vis-à-vis des fonctions standard.

Le compilateur C utilisé par l'auteur autorise l'appel *en C* de 99 fonctions C standard (annexe 1), de 93 fonctions système (annexe 4), de 96 fonctions VDI et de 66 fonctions AES (annexe 2), soit un total de plus de 350 fonctions ! Par comparaison le nombre d'instructions C est presque ridicule (32) et laisse deviner l'importance d'une bonne connaissance des bibliothèques, sans lesquelles il n'y aurait tout simplement pas de programme C.

2.4.5 Erreurs de documentation

Ne faites jamais une totale confiance dans la documentation. Sans parler de certains ouvrages où l'accumulation d'erreurs laisse pantois, même la documentation livrée avec votre compilateur C doit être remise en cause. Vous trouverez d'ailleurs, au fil de cet ouvrage, nombre de correctifs apportés à celle-ci

mais, même si en écrivant et en relisant ce livre, nous avons tenté de traquer la moindre erreur, l'infaillibilité n'est pas de ce monde. Seule votre expérience est un guide solide en la matière.

Lorsque l'on constate en ouvrant la documentation sur le GEM que les pages de départ sont consacrées au double système de coordonnées NDC et RC alors que *seul le système de coordonnées RC est implanté sur le 520ST,* on doit devenir soupçonneux. Lorsqu'on s'aperçoit que certaines instructions VDI (v_cellaray ou vrq_valuator par exemple) sont prises en compte par le compilateur mais non exécutées, on devient logiquement prudent !

Plus encore que les erreurs, ce sont les manques dans la documentation qui conduisent à des programmes apparemment parfaits mais parfaitement inutilisables sinon pervers. En ce domaine encore l'expérience est la seule règle, surtout lorsqu'on profite de l'expérience des autres ! L'étude de listings de programmes même courts (surtout courts, si vous débutez) est hautement profitable. Si l'auteur n'est pas lui-même un débutant, la moindre tournure étonnante doit en être examinée, elle correspond souvent à la résolution d'un problème et même si l'auteur ne saurait expliquer précisément pourquoi ça marche ainsi, l'essentiel est bien que ça marche ! Le pragmatisme est de rigueur en programmation.

Parmi quelques manques, signalons par exemple que la modification de hauteur de caractères en GEM (vst_height) ne fonctionne en hauteur qu'avec des multiples de la hauteur de police mais que les valeurs intermédiaires se traduisent par une modification de *la largeur.* Notons aussi que les attributs de remplissage n'ont aucune influence sur le tracé de caractère, ainsi une couleur de remplissage noire et une couleur de texte blanche sur un écran blanc conduisent à des caractères invisibles, car blanc sur blanc !

Enfin, certaines règles non définies rendent un programme plus sûr : appeler la fonction « wind_update(1) » avant de modifier l'écran, effacer la souris avant un affichage, réserver une zone mémoire pour stocker le contenu de l'écran et le récupérer pour retraçage, ne jamais réserver toute la mémoire disponible mais en laisser au moins 4Ko pour le système (cf. fsel_input( )), etc., selon le type de programme que vous désirez faire.

3. PROGRAMMATION EN ASSEMBLEUR

Les instructions et les modes d'adressage du 68000 seront supposés connus du lecteur, leur étude pouvant être effectuée à partir de l'un des ouvrages signalés dans la bibliographie. L'annexe 5 fournit un récapitulatif des instructions avec les modes d'adressage admis et l'incidence sur le registre d'état, simple aide-mémoire dont la consultation pourrait se révéler utile lors de la programmation ou de la lecture de ce chapitre.

3.1. *Méthodes de programmation*

La programmation en assembleur 68000, de par l'étendue du registre d'instructions et des modes d'adressage, est un régal ! Elle se rapproche d'ailleurs considérablement de la programmation en C dès lors que l'on dispose d'un macro-assembleur (ce que n'est pas l'assembleur de Digital Research), tout en autorisant de nombreuses fonctions système et quelques fonctions GEM supplémentaires, non connues du compilateur C.

Le passage d'un programme assemblé à travers l'éditeur de liens permet en effet de bénéficier de toutes les bibliothèques système. Il est donc possible d'appeler en assembleur, par exemple, la fonction VDI vclrwk (effacement d'écran) par :

```
move.w  identif,-(a7)
jsr     _vclrwk
addq    #2,a7
```

Ces trois lignes appellent plusieurs commentaires :

1) la variable « identif » représente l'identificateur d'application renvoyé lors de l'ouverture de celle-ci (voir v_opnvwk, Annexe 2). Elle correspond donc à une donnée initialisée en assembleur dans « .data » par :

   identif ds.w 1

On notera que l'appel de « identif » dans un programme permet de lire ou écrire *le contenu* de cette variable, alors que l'appel de « # identif » permet de lire *l'adresse* de cette variable.

2) L'instruction GEM vclrwk est précédée d'un trait de soulignement. *Ce trait de soulignement doit OBLIGATOIREMENT précéder tous les labels et noms de variables externes au programme,* donc faisant référence à un autre programme (en C ou en assembleur) ou à des fonctions GEM.

3) L'une des particularités de l'assembleur tient à la gestion de la pile, qui se traduit en l'occurrence par la dernière instruction de mise à jour. Cet aspect sera developpé plus loin.

Le langage assembleur 68000 est un langage simple, les opérations élémentaires réalisées étant aisément compréhensibles (lecture ou écriture de registre ou de mémoire, opérations logiques (ET, OU, comparaison, test, ect.), opérations arithmétiques (addition, multiplication, décalages, ...). Les difficultés naissent de certaines instructions sophistiquées, de particularités inhérentes à quelques instructions, de l'utilisation de fonctions externes (système ou GEM) et de la nécessité de structuration du programme lui-même. Aussi aborderons-nous chacune de ces difficultés dans les paragraphes suivants.

3.1.1. Instructions sophistiquées

Instructions utilisant le bit d'extension X (ADDX, SUBX et NEGX).

Le programmeur formé à l'assembleur 6800 ou 6809 se sera sans doute étonné de l'irruption de ce nouveau drapeau du registre d'état : l'indicateur d'extension X. L'intérêt de ce drapeau, proche de l'indicateur de retenue C, tient à ce qu'il n'est positionné que par un nombre très réduit d'opérations : les additions, les soustractions, les négations et les décalages logiques et arithmétiques. Il n'est pas modifié par les transferts (move), les opérations logiques et les multiplications et divisions, contrairement au bit de retenue. Il constitue donc un indicateur de base fort utile, en particulier pour des boucles associant opérations arithmétiques et transferts ou tests.

L'instruction NEGX apporte une possibilité supplémentaire, en autorisant la prise en compte de la retenue dans le positionnement d'un drapeau. Ainsi :

asl.w # 1,d0
negx.l d1

[illegible] dans D1 l'inverse de son contenu si le bit 15 de D0 était à 0 et placera [illegible] D1 l'inverse moins un de son contenu si le bit 15 de D0 était à 1 (si [illegible] nul, il sera respectivement égal à 0 ou −1, soit $FFFFFFFF).

Le bit d'extension X a donc une fonction proche du bit de retenue C [illegible] l'addition et la soustraction (cf. ADC et SBC de l'assembleur 6800 et [illegible] ADC et SBB de la famille 8080). Il offre des possibilités inédites en [illegible] de négation d'opérande.

[illegible] DBcc. Boucle avec branchement conditionnel

Sans correspondant parmi les instructions des processeurs 8 bits, l'ensemble des instructions DBcc du 68000 ne saurait être comparé qu'aux LOOP, [illegible], LOOPZ, LOOPNE et LOOPNZ du 8086, instructions très primaires [illegible] DBcc ! Nous n'aborderons pas ici la famille d'instructions de branchement conditionnel (BRA, BEQ, BGE, BHI, etc.) supposée connue et d'utilisation aisée. La famille d'instruction DBcc en constitue une extension, à [illegible] détente. Sa syntaxe générale est la suivante :

DBcc	Dn,	déplacement
test de condition	compteur de boucles	depl. sur 16 bits

Lorsque le processeur exécute cette instruction, il teste tout d'abord la condition. Si celle-ci est *vraie*, il passe à l'instruction suivante. Sinon, le registre de donnée Dn est décrémenté de 1. Si ce registre est alors égal à −1, le processeur passe à l'instruction suivante. Sinon il y a branchement à l'instruction pointée par le déplacement. Exemple :

```
        moveq   #7,d0
boucle
        move.b  (a0)+,(a1)+
        dbeq    d0,boucle
        ...
```

Cette courte séquence effectue un transfert d'au maximum huit octets de l'emplacement pointé par A0 vers l'emplacement pointé par A1. Si l'un des octets transférés est nul, il y a arrêt du transfert.

L'expérience montre que les erreurs d'utilisation de cette instruction proviennent du test de condition. Le branchement n'est effectué que SI LA CONDITION EST FAUSSE et le registre Dn différent de −1 ($FFFF). Les conditions de programmation viennent de ce que la plupart des assembleurs acceptent le code d'instruction DBRA à la place de DBF. Or l'instruction DBRA

n'existe pas et n'a aucun intérêt puisque la condition BRA est toujours vraie et qu'en conséquence il n'y aurait JAMAIS branchement alors qu'avec DBF il y a toujours branchement SI le registre Dn est différent de −1.

Instruction Scc. Test de condition et positionnement de drapeau

Cette instruction est sans doute l'une des plus méconnues du 68000, à tel point que la plupart des compilateurs C l'ignorent... Sa syntaxe est simple :

Scc	adresse
condition testée	adresse effective

Si la condition est VRAIE, l'octet pointé par l'adresse est mis à $FF. Si la condition est FAUSSE, cet octet est mis à 0. Si l'adresse effective correspond à un registre, seul l'octet faible est positionné. L'intérêt de cette instruction vient de ce qu'elle permet de positionner simplement un sémaphore (variable booléenne).

Les erreurs de programmation liée à l'emploi de Scc proviennent avant tout du positionnement effectué ($FF si vrai, 0 si faux), différent du couple booléen classique 0 et 1. Elles proviennent également d'une utilisation détournée de ce test par le biais des instructions ST (SBRA est admis par certains assembleurs) et SF. Dans ces deux cas, il n'y a aucun test mais simple positionnement à $FF pour ST et à 0 pour SF. Ce détournement d'instruction est à déconseiller, il n'offre pas de gain de rapidité d'exécution et rend le programme moins lisible.

Instruction MOVEP. Transfert vers des mémoires de périphériques

L'instruction MOVEP sert en principe à transférer des données vers des registres de périphériques. Lorsque les circuits de gestion de ces périphériques fonctionnent en 8 bits, il est en effet fréquent que les adresses de registres correspondent à des octets, soit tous d'adresses paires, soit tous d'adresses impaires. L'instruction MOVEP permet de transférer des octets se suivant (dans un registre ou des mémoires) vers des adresses toutes paires ou impaires. Exemple :

```
lea      $FFFFFA01,A0
moveq    #$0,D0
movep.l  D0,0(a0)
```

(exemple tiré de la routine d'initialisation du 68901, voir Annexe 3). Cette séquence place 0 en $FFFFFA01, $FFFFFA03, $FFFFFA05 et $FFFFFA07.

[illegible] l'adresse contenue dans A0 avait été paire, ce sont les octets d'adresse paire qui [illegible] aient été modifiés.

En dehors d'un transfert rapide et économique vers les registres d'entrée-sortie, l'instruction MOVEP peut être utilisée par exemple pour provoquer des effets graphiques. La plupart des routines de scrolling horizontal sont [illegible] fondées sur MOVEP...

[illegible]ion LINK et UNLK. Allocation temporaire de mémoire

Les instructions LINK et UNLK sont sans doute les instructions les plus [illegible]ées par les compilateurs (après MOVE tout de même)) et les moins fré[illegible]s dans les programmes assembleur ! Il s'agit d'instructions de haut [illegible] (l'envers du RISC en quelque sorte), spécifiques au 68000 et autori[illegible] l'emploi d'un registre d'adresse (en principe A6) comme pointeur de [illegible], autrement dit comme une sorte de second pointeur de pile. La [illegible]xe de LINK est la suivante :

LINK An , déplacement négatif

[illegible] An désigne un registre d'adresse.

A l'exécution de cette instruction, le processeur empile le contenu actuel de An (que nous fixerons ici à A6, comme dans tout le système TOS et GEM), [illegible] décrémente le pointeur de pile d'une valeur égale au déplacement. Sup[illegible] par exemple que A6 = $77E00 et A7 =$7FFF8, l'instruction :

LINK A6, $FFF0

[illegible] après exécution par : A6 = $7FFF4 et A7 = $7FFE4. Le contenu des [illegible]moires $7FFE4 à $7FFF8 sera le suivant :

$7FFE4.L quelconque (haut de la nouvelle pile)
$7FFE8.L quelconque
$7FFEC.L quelconque
$7FFF0.L quelconque
$7FFF4.L = $77E00 (ancien contenu de A6)
$7FFF8.L = ancien haut de la pile (souvent adresse de retour)

On dispose ainsi de 16 octets (de $7FFFE4 à $7FFFF3) disponibles pour [illegible] les variables locales de la routine par exemple. Ainsi « move.w # 6,$fff0(a6) » placera 0 en $7FFE4 et 6 en $7FFE5, etc. Le registre A6 [illegible] ainsi un pointeur vers la zone des variables locales (avec un déplace[illegible] négatif) ET vers le contenu de l'ancienne pile (paramètres envoyés à la [illegible]ine par exemple).

L'instruction corollaire « UNLK An » placée en fin de routine permet de [illegible] dans le registre An et le pointeur de pile A7 leurs contenus lors de l'appel de LINK. Dans l'exemple ci-dessus, on aura donc A6 = $77E00 et A7 =$7FFF8 après UNLK.

3.1.2. Particularités de certaines instructions

Le 68000 se distingue par la linéarité de son jeu d'instructions et de modes d'adressage. La plupart des modes d'adressage sont autorisés avec les instructions principales et des instructions proches traitent leurs opérandes de façon identique. Quelques exceptions existent pourtant qui, naturellement, occasionnent de fréquentes erreurs de programmation. Nous traiterons de ces exceptions après avoir rappelé quelques règles de bon sens :

— Toute instruction traitant un registre d'adresse doit traiter celui-ci sur un MOT ou un LONG MOT, jamais sur un octet.
— Toute adresse d'opérande mot ou long mot doit être PAIRE. Curieusement le Basic fourni avec le 520 ST transgresse cette règle en autorisant « POKE » sur un mot ou un long mot accompagné d'une adresse impaire (le transfert est effectué octet par octet). Mauvaise habitude à ne pas prendre !
— Tout transfert d'une valeur dans un registre d'adresse modifie TOUT ce registre. Si la valeur est un entier dont le bit fort est à 0, alors le mot fort du registre d'adresse est mis à 0000. Sinon il est mis à $FFFF.
— La pré-incrémentation et la post-incrémentation utilisent comme incrément ou décrément le nombre d'octets de l'opérande (1 si octet, 2 si mot, 4 si long mot). Ainsi « move.w (A1)+,D0 » provoquera, après transfert du mot contenu dans (A1) dans D0 une incrémentation de 2 du registre A1.

EXCEPTION : Tout empilement d'un octet dans la pile A7 provoque une décrémentation de 2 du pointeur de pile. Exemple :

move.b # 8,-(a7)

décrémente de 2 la pile et place $0008 sur la pile.

— Dans l'instruction « DBRA Dn,déplacement », le nombre de branchements effectués est égal au contenu de Dn PLUS UN, puisqu'il a bouclé tant que Dn est différent de -1.
— Le compteur de programme (PC) pointe toujours après le CODE de l'instruction. Ce code étant toujours de 2 octets, PC pointe 2 octets après le début de l'instruction.
— L'instruction DIVS *n'est pas exécutée* si le quotient obtenu par la division ne tient pas sur 16 bits. Dans ce cas toutefois, le drapeau V du registre d'état est mis à 1. Cette règle est aussi valable pour DIVU.
— Les instructions arithmétiques ADD ou SUB autorisent 4 modes d'adressage :

<adresse effective>	registre de données Dn
registre de données Dn	<adresse effective>
<adresse effective>	registre d'adresse An
valeur immédiate	<adresse effective>

Par contre l'instruction CMP *n'autorise pas le mode d'adresse « registre de données <adresse effective> »*. Autrement dit, une comparaison entre un registre de données et une adresse effective (registre, mémoire) doit TOUJOURS être effectuée avec le registre de données en deuxième opérande.

— Les instructions logiques AND ou OR autorisent 3 modes d'adressage :

<adresse effective>	registre de données Dn
registre de données Dn	<adresse effective>
valeur immédiate	<adresse effective>

Par contre l'instruction EOR *n'autorise pas le mode d'adressage « <adresse effective> registre de données »*. C'est-à-dire qu'un ou exclusif ne peut être effectué qu'entre un registre de données et une adresse effective (registre, mémoire). Ces deux exceptions (CMP et EOR) tiennent au partage de la « type B » par ces deux instructions, une certaine incompatibilité de codes qui conduit à ces restrictions bien gênantes.

— L'instruction EXG, d'échange du contenu de deux registres, peut s'appliquer à deux registres de données, à deux registres d'adresses ou à un registre de donnée et un registre d'adresse, autrement dit l'instruction EXG Ax,Dy est illégale (certains compilateurs l'acceptent toutefois, en la convertissant en EXG Dy,Ax).

On pourra enfin regretter l'absence de certaines instructions ou options en assembleur 68000. Ainsi l'instruction SWAP ne permet d'échanger que des MOTS d'un registre de données. Elle ne peut donc s'appliquer à une mémoire et surtout ne permet pas d'échanger des octets. La mise à 0 ou à 1 des drapeaux du registre d'état nécessite par ailleurs l'utilisation d'un opérateur logique AND ou OR avec le registre d'état. Pas de CLC comme en assembleur 6809 ou 8086 ! L'absence d'une instruction DDA d'ajustement décimal fait également défaut mais il faut reconnaître que le jeu d'instructions 68000 est l'un des plus complets et des plus souples d'emploi.

3.1.3. Utilisation de fonctions externes en assembleur

Les programmes fournis en 3.2 et 3.3 illustrent le mode d'emploi normal des fonctions système (TOS et GEM) en assembleur. Ce fonctionnement est simple : empilement des paramètres, appel de la fonction, remise à jour de la pile. Pour utiliser une fonction dont on connaît les paramètres C, on empilera d'abord le dernier paramètre, puis le précédent, et ainsi de suite jusqu'au premier paramètre. Une grande attention est de rigueur quant à la longueur et au nombre des paramètres, aucune vérification n'étant effectuée par l'éditeur de liens (cf. 4.1.).

L'un des intérêts de l'assembleur tient à la possibilité d'appeler des fonctions non référenciées dans l'éditeur de liens (par exemple ESCAPE 101 ou ESCAPE 102, voir annexe 2). Là commencent les difficultés... bien que celles-ci soient fort surmontables. La solution réside dans l'utilisation de TRAP 2. Toutes les fonctions GEM se ramènent en effet à l'exécution d'un TRAP 2, avec $73 dans D0.W pour les fonctions VDI et avec $C8 dans D0.W pour les fonctions AES.

Pour le stockage des paramètres d'appel on utilisera les variables de SON programme « contrl », « intin », « ptsin », « intout » et « ptsout » et l'on initialisera une table d'adresses pointant ces 5 zones de paramètres. L'adresse de cette table sera placée dans le registre D1.L avant l'appel de la fonction.

Exemple : Appel de la fonction ESCAPE 102 initialisant une police de caractères en mode alphanumérique.

```
.text
    ...         identif,12(a0)
    lea         _contrl,a0          pointe les variables de contrôle
    move.w      # 5,(a0)            code général de la fonction ESCAPE
    clr.w       2(a0)               contrl (1)=0; nbre vecteurs entrée
    clr.w       4(a0)               contrl(2)=0, nbre vecteurs sortie
    move.w      # 2,6(a0)           contrl(3)=2, long. zone entrée
    clr.w       8(a0)               contrl(4)=0, long. zone sortie
    move.w      # 102,10(a0)        contrl(5)=102, code de ESCAPE 102 !
    move.w      identif,12(a0)      contrl(6)= identif. d'application
    lea         _intin,a0           pointe la zone des param. d'entrée
    move.l      # police,(a0)       intin(0-1) = adresse police caract.
    move.l      # table,dl          d1= adresse table de pointeurs
    move.w      $73,d0              d0 =$73, code des fonctions VDI
    trap        # 2                 exécute la fonction
    rts
    ...
.data
table           dc.1        # _contrl, # _intin, # _ptsin, # _intout, # _ptsout
_contrl  ds.w       12
_intin   ds.w       128
_ptsin   ds.w       256
_intout  ds.w       128
_ptsout  ds.w       12
identif  ds.w       1
police   ...        .......         (en-tête et données de la police)
  ...
```

La fonction vst_load_fonts permet également d'illustrer l'appel de fonctions VDI en assembleur mais de façon sensiblement différente puisque cette fonction a un nom à l'édition de liens. Malheureusement trois paramètres indispensables à son exécution ne figurent pas dans les paramètres d'appel : CONTRL(7), CONTRL(8-9) et CONTRL(10-11). Pour placer ces paramètres avant l'appel, on pourra utiliser la séquence suivante :

```
link    a6, # $fffe          pour réserver 2 octets
dc.w    $A000                appel de la ligne « A » pour qu'A0
movea.l 4(a0),a1             pointe bloc de variables
movea.l (a1),a1              A1 = adresse de CONTRL
move.l  # tampeff,14(a1)     CONTRL(7-8) = adr. tampon d'effets
move.w  depeff,18(a1)        CONTRL(9) = deplac. tampon
                             échelle
move.l  # police,20(a1)      CONTRL(10-11) = adr. de la police
move.w  # 0,(a7)             « select », toujours nul
move.w  # identif, -(a7)     identificateur de l'application
jsr     _vst_load_fonts      exécution de la fonction
addq.l  # 4,a7               mise à jour de la pile
unlk    a6                   pour restaurer A6 et A7
rts
```

En théorie l'instruction de mise à jour de la pile est inutile avant l'instruction UNLK qui, de toutes façons, remet la pile dans son état lors de l'appel de LINK mais il est préférable de la conserver car qui sait si vous ne rajoutez pas des instructions entre « addq.l 4,a7 » et « unlk a6 » ? Auquel cas la non remise à jour de la pile pourrait provoquer quelques surprises !

L'appel de fonctions AES s'effectue de la même manière mais avec D0=$C8 et avec D1.L égal à l'adresse d'une table d'adresses des zones de stockage des variables de base de l'AES. Rappelons que ces variables de base sont dans l'ordre : _control, _global, _int_in, _int_out, _addr_in, _addr_out. Les zones de variables sont naturellement distinctes des zones de variables de VDI. Quoiqu'il en soit, toutes les fonctions AES étant connues du compilateur, leur appel s'effectuera normalement par un simple jsr (ex : jsr _graf_mouse), après empilement des paramètres.

L'appel de fonctions système TOS, passant par l'emploi de TRAP 1, TRAP 13 ou TRAP 14, est largement détaillé en Annexe 4. Il est extrêmement simple en assembleur et de nombreux exemples sont fournis dans le programme d'initialisation d'un disque virtuel.

3.1.4. Structuration des programmes

La structuration est un peu la tarte à la crème des programmeurs. Ses partisans sont légion mais la lecture de nombreux listings prouve que les intentions les meilleures conduisent aux embrouillaminis les plus extravagants et les moins lisibles... L'expérience montre que mieux vaut passer quelques heures à réfléchir au meilleur et au plus simple algorithme plutôt que de tracer soigneusement les losanges et les rectangles d'un ordinogramme dont il est pra-

tiquement certain que l'on s'éloignera au fil des corrections, des ajouts et des tests. Certaines règles générales nous paraissent cependant ne pas devoir être transgressées en langage assembleur (ni, le plus souvent, en langage C) :

* La partie la plus difficile ou la plus délicate, voilà par quoi commencer le programme. A quoi servirait de passer des jours entiers à programmer le plus simple si vous ne pouvez venir à bout du plus dur ?
* Excepté pour les variables système ou l'adressage des circuits périphériques, n'utilisez JAMAIS d'adresses fixes. Faites comme s'il était impossible d'utiliser l'adressage direct. Avec le temps vous éviterez même l'emploi de tout nombre dans la partie programme en définissant les valeurs par l'instruction assembleur « equ ». Cette règle est transgressée dans les exemples de programmes de ce chapitre pour des raisons de place...
* Une routine destinée à être appelée par un module extérieur doit être précédée d'un trait de soulignement. Évitez par contre de précéder de ce trait les labels et noms de variables uniquement à usage interne.
* Seuls les 8 premiers caractères d'un label sont pris en compte par l'assembleur. Il est donc préférable d'identifier des routines ou des variables proches plutôt par un suffixe que par un préfixe, contrairement à une tendance dominante... N'hésitez pas à utiliser les nombres dans les labels (pas comme premier signe naturellement).
* L'anglais est la langue de l'informatique par excellence mais, à moins d'être tombé dans la Tamise quand on était petit, est-il vraiment nécessaire de baptiser « handle » un identificateur ou « screen » un écran ? Les noms les plus usités sont encore les plus parlants et aident à la correction d'erreurs.
* Toute nouvelle variable appelée dans un programme doit IMMÉDIATEMENT être enregistrée dans la zone des données et répertoriée en tête du programme comme « .globl » si elle doit être utilisée par un autre module.
* Deux variables qui se suivent dans la liste des « .data » ne se suivront sans doute pas lors de l'exécution. Tout tableau doit donc constituer un bloc.
* Une variable chaîne doit toujours être suivie d'un octet nul et d'une instruction assembleur « .even » rétablissant la parité d'adressage.
* La tendance « naturelle » d'une boucle est d'être infinie. Vérifiez toujours que la ou les condition(s) de sortie seront validées quel que soit l'entrée dans cette boucle.
* Lors d'une routine de choix avec exécution de sous-programme selon le terme du choix, méfiez-vous toujours des choix non prévus, des débordements de limites en particulier.
* Mieux vaut empiler trop de registres que pas assez ! Tout appel de sous-programme devrait être précédé d'une sauvegarde des registres D0 à D7

A0 à A5, avec restitution de ces registres au retour, sauf cas particuliers et maîtrise complète de l'assembleur. Mieux vaut ajouter des mémoires de stockage temporaire de résultats que risquer de détruire le contenu d'un registre.

- La pile est la meilleure et la pire des choses ! Indispensable pour le passage des paramètres et la sauvegarde, elle est aussi à l'origine de nombreux « plantages ». Signalez vos empilements en marge et vérifiez les dépilements. Plus le dépilement est proche de l'empilement, mieux c'est !
- En assembleur 68000, plus de la moitié des erreurs commises et non décelées à l'assemblage proviennent de la confusion entre OCTET et MOT, MOT et LONG MOT. Mieux vaut remettre un registre de données à 0 inutilement que conserver une valeur résiduelle dans le mot fort qui viendra, parfois dans une circonstance tout à fait exceptionnelle, interférer sur tel ou tel résultat. Utilisez le plus souvent possible le suffixe .W pour une instruction portant sur un entier bien qu'il s'agisse du suffixe par défaut, cela vous aidera à repérer les erreurs de taille d'opérande. Rappelez-vous que MOVEQ traite toujours un long mot même si vous le faites suivre de .W !
- Plus une tâche peut être divisée en multiples sous-tâches, mieux cela vaut ! Il est ainsi possible de réutiliser une même routine pour des objectifs parfois totalement étrangers et, surtout, la correction des erreurs en est facilitée.
- N'utilisez que les instructions et les modes d'adressage dont vous avez parfaitement saisi le fonctionnement et les conséquences.
- Le moniteur (SID pour l'assembleur Digital Research) est le meilleur et le seul ami du programmeur. Comme le testeur de l'électronicien, il constitue l'outil par excellence du « deboguage » et son utilisation doit donc être aussi bien connue que celle de l'assembleur.
- L'entrée-sortie utilisateur est le calvaire du programmeur ! Tests, tests, tests... qu'ajouter sinon que c'est toujours l'entrée au clavier incongrue et totalement « imprévisible » que choisira l'utilisateur... et qui plantera votre programme.
- Mieux vaut prendre l'utilisateur de votre programme pour un imbécile ! Ce qui vous paraît évident, c'est ce qui n'est pas évident... La preuve : reprenez un programme écrit il y a quelques années et essayer de l'utiliser sans mode d'emploi !
- La récursivite est rarement indispensable et toujours à éviter. Combien de processeurs se sont perdus dans ses boucles...
- Les algorithmes les plus ringards sont souvent les plus sûrs ! Pourquoi vouloir utiliser un algorithme récursif de tri rapide quand un Shell-Metzner ou même un tri à bulle fait correctement les choses à moindres frais, intellectuels s'entend.

* Un programme mal conçu et mal écrit est beaucoup plus difficile à protéger qu'un programme clair. Confusion n'implique pas complexité.
* Pourquoi passer tant de temps à rechercher la preuve de validité d'un algorithme s'il fonctionne (on serait tenté de dire : sa preuve, c'est qu'il tourne !). La fiabilité de la preuve par 9 est totale mais qui l'utilise pour une multiplication ? Or il n'existe pas de preuve par 9 en programmation !
* L'optimisation d'un programme est le meilleur moyen de le détruire ! Optimisez les rares séquences ou la rapidité est vraiment indispensable ou utile (boucles à fort taux de répétition, entrée-sorties disque, etc.) mais ne pratiquez pas du retouchage... La chirurgie esthétique fait très mauvais ménage avec la programmation.

Les assertions précédentes n'engagent naturellement que leur auteur et chacun est libre de suivre sa voie. Elles constituent néanmoins les fruits d'une expérience de programmation, ponctuée d'échecs amers mais aussi de quelques satisfactions, dûes souvent au respect de ces règles.

3.2. Exemple de programme : un disque virtuel

Le concept de disque virtuel, directement issu de la « grosse » informatique, est encore peu usité en matière de micro-ordinateurs. Par une rapide paraphrase, on pourrait définir le disque virtuel comme une portion de mémoire considérée par le système non comme une part de la mémoire vive mais comme un périphérique comme les autres, dans lequel il est possible d'effectuer des lectures et des écritures.

Si la notion de disque virtuel est encore peu répandue, c'est bien à cause de son inutilité sur les micro-ordinateurs, jusqu'au 520 ST... Lorsque la mémoire vive suffit à peine à contenir un programme, à quoi servirait un disque virtuel ? Avec le 520 et le 1040 ST, ce dernier modèle étant celui sur lequel a été écrit cet ouvrage, le disque virtuel devient par contre très intéressant, sinon indispensable.

L'atout maître du disque virtuel c'est naturellement sa rapidité d'accès, considérablement plus grande que celle d'un lecteur de disquette ou même de disque dur. Le revers de la médaille, c'est naturellement sa volatilité (une coupure de courant et le disque virtuel est mort...) et son caractère éphémère puisqu'il nécessite une sauvegarde sur disque physique avant chaque extinction de la machine. Reste que pour des applications nécessitant de fréquents accès disques comme des compilations ou des traitements de fichiers, il autorise un confort et une rapidité sans concurrence.

Le programme listé et commenté dans les pages suivantes est un programme TOS, c'est-à-dire qu'il n'utilise aucunement le GEM, contrairement au programme présenté en 3.3. Il est structuré en deux parties distinctes : la première, de bdpdur au premier ensemble de variables, est constituée des routines de test du numéro de drive et des routines d'écriture et de lecture du disque virtuel (appréhendé ici comme un disque dur à accès DMA, cf. routines cherdur, lecsec et écrisec) ; la deuxième, de initdur à la fin , est constituée de la routine maître (routprin), de la routine d'initialisation du disque (initdur) et des diverses routines de communication avec l'utilisateur.

Ce disque virtuel possède un bloc catalogue de 18 secteurs (9 groupes comme pour les disques physiques, cf. Annexe 4, trap 1, fonction $36). Ce bloc amputé la mémoire de 9 Ko mais il semble ne pas pouvoir être restreint dans l'état actuel du système... Naturellement un disque virtuel n'a pas à être formaté, le bloc catalogue étant initialisé par la routine « initboot ». Toute tentative de formatage se traduirait d'ailleurs par un « plantage » du système qui effectuerait des actes *physiques* aux disques A et B avec les résultats que l'on devine !

Ce programme, une fois entré à l'aide d'un éditeur, sera assemblé, lié avec lui-même (!) et rendu exécutable, ce qui correspond aux commandes suivantes avec le compilateur Digital Research (le programme source étant arbitrairement nommé « disqvirt ») :

```
asm68 -f b: -u -l b:disqvirt.s
link68 [item[b:],u,s] b:disqvirt.68k=b:disqvirt
relmod b:disqvirt
```

Le fichier objet final « disqvirt.prg » sera converti en fichier TOS par la commande d'installation d'application du Bureau (menu « Options »). Il doit comporter 2047 octets. L'édition de liens du programme est assez artificielle, elle est destinée à produire le fichier intermédiaire utilisé par «relmod ». Quoi qu'il en soit, le résultat final est strictement identique au désassemblage avec le listing ci-dessus, les impératifs de relogeabilité étant naturellement pris en compte.

Au lancement, ce programme, contrairement à certains programmes de disque virtuel commercialisés, indique la capacité mémoire maximale disponible. Il est fort déconseillé de consacrer tout cet espace au disque virtuel... mais l'habitude et les échecs vous conduiront rapidement à savoir combien de mémoire doit être laissée à tel ou tel programme. A titre indicatif, un éditeur comme « Mince » nécessite environ 140 Ko, de même qu'une compilation ou une édition de liens. Une simple soustraction vous permettra donc de savoir combien de mémoire vous pouvez laisser au disque virtuel.

Au retour du programme, il est recommandé de cliquer une icone quelconque et d'appeler l'option du menu « Installer un disque », faute de quoi la fenêtre correspondant au disque ne pourra pas être visualisée. On notera cependant que cela n'est pas obligatoire et peut être effectué à tout moment. On relèvera aussi qu'il est possible de créer plusieurs disques virtuels, en appelant plusieurs fois « disqvirt.prg », mais l'intérêt réel d'une telle potentialité n'apparaîtra qu'une fois le système d'exploitation en ROM et avec un 1040 ou même 2080 ST... Enfin, notons qu'il est impossible de jeter le disque virtuel, de même que pour les autres icones.

Ce programme n'est pas un modèle de programmation, il a été conçu essentiellement dans un but utilitaire et pédagogique. L'optimisation du code a été recherchée en ce qui concerne les routines de lecture et d'écriture en mémoire (lecsec et écrisec), la routine transec pouvant d'ailleurs être encore optimisée... en alignant 512 fois l'instruction « move.b (a1)+, (a0)+ », mais au détriment de la mémoire disponible. Car, bien sûr, les routines de test, de lecture et d'écriture en mémoire, ne sont pas effacées après l'exécution de «disqvirt.prg » puisqu'elles servent à la gestion du disque virtuel !

```
*************************************************************
*                                                           *
*                          dvirtuel.s                       *
*                                                           *
*        disque virtuel avec choix de la taille memoire     *
*        laissee a l'application et de l'espace memoire     *
*        affecte au disque virtuel.                         *
*                                                           *
*************************************************************
*
.text
_main
        bra       routprin          * vers programme d'installation  *
bdpdur
        move.w    4(a7),d0          * routine de pointage du bloc de *
        movea.l   vieubdp,a0        * parametres du disque virtuel   *
        lea       retbdp,a1         * ou d'un disque normal (adresse *
        bra       compnum           * conservee dans 'vieubdp'       *
lececdur
        move.w    $e(a7),d0         * routine de lecture ou ecriture *
        movea.l   vieulec,a0        * de secteur, soit du disque vir-*
        lea       cherdur,a1        * tuel (cherdur), soit du disque *
        bra       compnum           * normal (consevee dans vieulec) *
modedur
        move.w    4(a7),d0          * routine de demande du change-  *
        movea.l   vieumod,a0        * ment de disque eventuel, re-   *
        lea       retnul,a1         * tourne 0 si disque virtuel     *
compnum
        cmp.w     numdriv,d0        * compare identificateur disque  *
        bne       sautrout          * appele avec identificateur du  *
        movea.l   a1,a0             * disque virtuel (numdriv)       *
sautrout
        jmp       (a0)              * execute routine selon disque   *
retbdp
        move.l    #bdpara,d0        * retourne adresse bloc de para- *
        rts                         * metres du disque virtuel       *
```

```
retnul
        moveq   #0,d0           * retourne 0 car disque virtuel  *
        rts                     * ne peut avoir change...        *
erreur
        moveq   #0,d0           * place dans d0 le numero du     *
        move.w  $c(a7),d0       * secteur de depart a transferer *
        moveq.l #9,d1           * multiplie ce numero par 512    *
        asl.l   d1,d0           * (nombre d'octets par secteur)  *
        add.l   adrsto,d0       * met dans a0 (via d0) l'adresse *
        movea.l d0,a0           * de depart du bloc a transferer *
        movea.l 6(a7),a1        * a1 = pointeur vers tampon      *
        move.w  4(a7),d0        * d0 = 0 si lecture,1 si ecrit.  *
        tst.w   d0
        beq     lecsec          * si lecture -> lecsec           *
        cmpi.w  #1,d0
        beq     ecrisec         * si ecriture -> ecrisec         *
        bra     fintrans        * sinon retour avec erreur       *
lecsec
        exg     a0,a1           * echange source et destination  *
ecrisec
        move.w  10(a7),d0       * d0 = nombre de secteurs a lire *
        beq     retlecec        * ou ecrire (si nul -> fin)      *
transfert
        moveq   #$1f,d1         * compteur de boucle (32 fois 16)*


transec
        move.b  (a1)+,(a0)+     * Cette repetition de transferts *
        move.b  (a1)+,(a0)+     * se justifie par le gain de ra- *
        move.b  (a1)+,(a0)+     * pidite obtenu :                *
        move.b  (a1)+,(a0)+     * le transfert de 512 octets     *
        move.b  (a1)+,(a0)+     * s'effectue en 6484 cycles ma-  *
        move.b  (a1)+,(a0)+     * chine (soit 0.81 ms) ...       *
        move.b  (a1)+,(a0)+     * La boucle simple :             *
        move.b  (a1)+,(a0)+     *         move.w  #511,d1        *
        move.b  (a1)+,(a0)+     * transec                        *
        move.b  (a1)+,(a0)+     *         move.b  (a1)+,(a0)+    *
        move.b  (a1)+,(a0)+     *         dbf     d1,transec     *
        move.b  (a1)+,(a0)+     * produit un transfert de 512    *
        move.b  (a1)+,(a0)+     * octets en 11284 cycles, soit   *
        move.b  (a1)+,(a0)+     * 1.41 ms...                     *
        move.b  (a1)+,(a0)+     * Le gain de temps par secteur   *
        move.b  (a1)+,(a0)+     * n'est donc pas negligeable !   *
        dbf     d1,transec
        subq.w  #1,d0           * boucle pour secteur suivant    *
        bne     transfert       * jusque fin du transfert        *
retlecec
        moveq   #0,d0           * retour sans erreur             *
fintrans
        rts
*
bpara
        dc.w    $200            * bloc de parametres du disque   *
        dc.w    2               * virtuel: $200, soit 512 octets *
        dc.w    $400            * par secteur du disque, etc.    *
        dc.w    7               * parametres identiques a ceux   *
        dc.w    5               * d'un disque normal, sauf para- *
        dc.w    6               * metres de taille globale, note *
        dc.w    $12             * dans 'koallou' (nombre de Ko   *
koallou                         * disponibles sur disque virtuel)*
        dc.w    $2e ds 1
        dc.w    0 ds 1
numdriv                         * stockage du numero affecte au  *
        dc.w    0 ds 1          * disque virtuel                 *
octallou
        dc.l    0 ds 1          * stock nombre octets du disque  *
ptlesys
        dc.l    0 ds 1          * stock adresse page de base     *
        dc.l    0 ds 1
```

```
adrsto
        dc.l    0 ds 1          * stock adr.depart tampon disque *
respile
        dc.l    0 ds 1          * stockage temporaire de variable*
vieubdp                         * stock adresse routine demande  *
        dc.l    0 ds 1          * bloc de parametres disque norm.*
vieulec                         * stock adresse routine ecriture *
        dc.l    0 ds 1          * lecture secteur normale        *
vieumod                         * stock adresse routine de test  *
        dc.l    0 ds 1          * de changement de disque normal *
bourrage
        dc.l    0,0,0,0 ds 4    * quatre longs mots pour rien !  *
*
*                                 Initialise le disque virtuel   *
initdur
        clr.l   -(a7)
        move.w  #$20,-(a7)      * passage en mode superviseur    *
        trap    #1
        addq.l  #6,a7

        move.l  d0,respile      * preserve ptr pile utilisateur  *
        suba.l  a5,a5           * pour pointer debut RAM         *
        move.l  $4c2(a5),d0     * vecteur carte des disques      *
        or.l    masqdr,d0       * modifie pour tenir compte du   *
        move.l  d0,$4c2(a5)     * nouveau disque implante        *
        move.l  $472(a5),vieubdp * stocke adresses normales des  *
        move.l  $476(a5),vieulec * routines relatives aux disques *
        move.l  $47e(a5),vieumod
        move.l  #bdpdur,$472(a5) * remplace dans les variables   *
        move.l  #lececdur,$476(a5) * systeme par les nouvelles   *
        move.l  #modedur,$47e(a5) * adresses de routines (utilis.) *
        move.l  respile,-(a7)
        move.w  #$20,-(a7)      * retour en mode utilisateur     *
        trap    #1
        addq.l  #6,a7
        move.l  octallou,d0     * d0 = nombre octets disque virt.*
        moveq.l #9,d1           * divise par 512 pour obtenir le *
        lsr.l   d1,d0           * nombre de secteurs             *
        sub.l   #$12,d0         * - 18 secteurs de 'boot'        *
        lsr.l   #1,d0           * conversion en K octets         *
        move.w  d0,koallou      * stockage nombre de Ko dispon.  *
        lea     okalloc,a0      * pointe message de bon retour   *
        move.l  a0,-(a7)        * pour affichage                 *
        move.w  #9,-(a7)
        move.l  octallou,d0     * place dans message le nombre   *
        bsr     convko          * de Ko disponibles dans disque  *
        trap    #1              * Affichage du message de retour *
        addq.l  #6,a7
        bsr     initboot        * Initialisation du 'boot'       *
        bsr     retour          * Affichage du message de fin    *
        clr.w   -(a7)
        move.l  #initdur,d0     * Place dans d0 nombre d'octets  *
        sub.l   #bdpdur,d0      * a conserver apres fin du job   *
        add.l   #$104,d0        * soit routines disque virtuel   *
        move.l  d0,-(a7)        * + 256 octets de tampon         *
        move.w  #$31,-(a7)      * Termine l'implantation en      *
        trap    #1              * conservant octets precedants   *
*
routprin                  * Depart reel du programme        *
        movea.l a7,a5
        movea.l $4(a5),a5       * adresse de la page de base     *
        move.l  a5,pilesys      * sauvegardee dans pilesys       *
        move.l  $c(a5),d0
        add.l   $14(a5),d0      * d0 = adresse page de base +    *
        add.l   $1c(a5),d0      * longueur texte + long. donnees *
        add.l   #$300,d0        * + longueur pile + 768 octets   *
        move.l  d0,d1
        add.l   a5,d1           * d1 pointe sommet de la pile de *
        and.l   #$fffffffe,d1   * l'application (adresse paire)  *
```

```
        movea.l   d1,a7
        move.l    d0,-(a7)          * stocke longueur globale appl.  *
        move.l    a5,-(a7)          * stocke ancien pointeur de pile *
        clr.w     -(a7)
        move.w    #$4a,-(a7)        * realloue au GEM la memoire non *
        trap      #1                * utilisee par l'application     *
        adda.l    #12,a7
        lea       entete,a0         * Affichage du panneau d'en-tete *
        bsr       affiche
        clr.l     -(a7)
        move.w    #$20,-(a7)        * passage en mode superviseur    *
        trap      #1
        addq.l    #6,a7

        suba.l    a5,a5             * pour pointer le debut de RAM   *
        move.l    $4c2(a5),d1       * d1 = vecteur carte disques     *
        move.l    d1,respile        * sauvegarde dans respile        *
        move.l    d0,-(a7)
        move.w    #$20,-(a7)        * retour en mode utilisateur     *
        trap      #1
        addq.l    #6,a7
        lea       masqdr,a0         * pointeur du masque des disques *
        move.w    #2,numdriv        * numero de disque par defaut(C) *
        move.l    #4,(a0)           * masque correspondant (bit 2)   *
plusdisq
        move.l    respile,d0        * d0 = vecteur carte disques     *
        and.l     masqdr,d0         * Si numero de disque n'appar-   *
        beq       memdisq           * tient pas au vecteur carte     *
        move.l    (a0),d0           * saut en memdisq. sinon recom-  *
        lsl.l     #1,d0             * mence avec masque pour bits 3  *
        move.l    d0,(a0)           * a 8 (soit disques D a H)       *
        addq.w    #1,numdriv
        cmpi.w    #8,numdriv
        ble       numdisq
        lea       plusdisq,a0       * Si disques C a H existent,     *
        bra       messfin           * affiche message 'plusdisq'     *
memdisq
        move.w    numdriv,d0        * numero du disque virtuel       *
        add.b     #$41,d0           * + code ASCII de 'A'            *
        move.b    d0,okalloc+41     * stocke dans message creation   *
        move.l    #$ffffffff,-(a7)
        move.w    #$48,-(a7)        * Demande du nombre d'octets     *
        trap      #1                * disponibles en memoire         *
        addq.l    #6,a7             * (retourne dans d0)             *
        lea       allocdef,a0       * pointe message d'erreur si...  *
        cmp.l     #$3000,d0         * si moins de 12 Ko disponibles  *
        blt       messfin           * pas de disque virtuel ...      *
        sub.l     #$10000,d0        * - 64Ko pour valeur par defaut  *
        move.l    d0,octallou       * stocke comme nombre par defaut *
        add.l     #$10000,d0        * replace le nombre a sa valeur  *
        lea       combko,a0         * pour conversion ASCII decimal  *
        bsr       convko            * et affichage maxi reservable   *
ckcem
        lea       combko,a0         * Affiche demande du nombre de   *
        bsr       affiche           * Ko a reserver ...              *
        lea       chaine,a0         * pointe debut zone de stockage  *
        move.l    a0,-(a7)          * nombre de Ko demandes et l'i-  *
        move.b    #6,(a0)+          * nitialise avec des espaces     *
        move.b    #0,(a0)+
        move.w    #5,d1
stocesp
        move.b    #$20,(a0)+
        dbra      d1,stocesp
        move.w    #10,-(a7)         * Attente d'entree d'une chaine  *
        trap      #1                * qui sera stockee a partir de   *
        addq.l    #6,a7             * chaine, sur 6 caracteres maxi  *
        lea       retcha,a0
        bsr       affiche           * retour de chariot pour present.*
        lea       chaine+1,a0
```

```
        move.l    octallou,d0        * teste si chaine entree =        *
        tst.b     (a0)+              * simple retour de chariot        *
        beq       alloumem           * Si c'est le cas -> alloumem     *

        bsr       fin0               * Sinon conversion nombre de Ko   *
        lea       memtpetite,a0      * demandes en nombre hexadecimal  *
        cmp.w     #11,d0             * Si nombre demande <= 11 Ko      *
        bls       messfin            * message 'memtpetite'            *
        move.w    #10,d1             * Sinon, conversion du nombre de  *
        lsl.l     d1,d0              * Ko demande en nombre d'octets   *
        move.l    octallou,d1
        add.l     #$10000,d1         * d1=nombre d'octets disponibles  *
        cmp.l     d1,d0              * Si demande <= disponible        *
        bls       alloumem           *  -> alloumem                    *
        lea       memtgrande,a0      * Sinon affichage du message      *
        bsr       affiche            * 'memtgrande' et nouv. demande   *
        bra       okmem
alloumem                       * Alloue memoire disque virtuel  *
        move.l    d0,octallou
        move.l    octallou,-(a7)     * Reservation du nombre d'octets  *
        move.w    #$48,-(a7)         * demandes pour le disque virtuel*
        trap      #1
        addq.l    #6,a7
        move.l    d0,adrsto          * Stocke ce nombre dans adrsto    *
        tst.l     d0                 * Si pas erreur, installe disque  *
        bne       initdur
        movea.l   #allocdef,a0       * Sinon, affichage 'allocdef'     *
messfin
        bsr       affiche            * Affichage de 'pressez Return..'*
        bsr       retour
        clr.w     -(a7)              * et fin du processus sans reser-*
        trap      #1                 * vation de memoire               *
fin0
        moveq     #0,d0
convasc                              * Conversion de nombre en hexa.   *
        cmpi.b    #$39,(a0)
        bhi       fini               * Si chiffre n'a pas un code com-*
        cmpi.b    #$30,(a0)          * pris entre 0 et 9, fin boucle   *
        bcs       fini
        mulu      #10,d0             * multiplie ancien total par 10   *
        move.b    (a0)+,d1           * puis y ajoute le nouveau chif-  *
        andi.l    #$f,d1             * fre decimal                     *
        add.l     d1,d0
        bra       convasc            * et boucle jusque fin du nombre  *
fini
        rts
*
* Sous-programmes
*
affiche                              * Affichage de message a l'ecran  *
        move.l    a0,-(a7)
        move.w    #9,-(a7)
        trap      #1
        addq.l    #6,a7
        rts
convko                               * Conversion d'hexa en ASCII      *
        moveq.l   #10,d1             * d1 pour division par 1024       *
        lsr.l     d1,d0              * d0 = nombre de Ko               *
        adda.l    #5,a0              * a0 pointe fin de la chaine      *
        move.w    #4,d1              * nombre de chiffres pour boucle  *
nombsui
        ext.l     d0                 * Si plus de chiffre ,fin         *
        beq       fini
        divs      #10,d0             * sinon, divise nombre par 10     *
        swap      d0                 * d0.w = reste de la division     *
        move.b    d0,-(a0)           * stocke dans la chaine           *
        addi.b    #$30,(a0)          *  + $30 pour code ASCII          *
        swap      d0                 * d0.w = quotient ancienne div.   *
        dbra      d1,nombsui         * boucle jusque fin du nombre     *
```

```
        rts
[illegible]                    * Initialisation du 'boot'        *
        movea.l   adrsto,a0
        moveq.l   #17,d0                 * compteur de boucles (18)        *
[illegible]
        move.l    #$1ff,d1               * compteur de boucles (512)       *
[illegible]
        move.b    #0,(a0)+               * Initialise a 0 dix-huit blocs   *
        dbra      d1,octet               * de 512 octets, soit 9 Ko        *
        dbra      d0,bloc
        movea.l   adrsto,a0              * Pointe debut du 'boot'          *
        adda.l    #$200,a0               * + 512 (donc 2eme secteur)       *
        move.l    #$f9fff£00,(a0)        * stocke 'F9FFFF' en debut de     *
        movea.l   adrsto,a0              * deuxieme secteur puis en debut  *
        adda.l    #$c00,a0               * de sixieme secteur              *
        move.l    #$f9ffff00,(a0)        * (pointeurs de 'boot')           *
        rts
[illegible]
        lea       pressret,a0            * Affichage du message de fin     *
        bsr       affiche                * affiche 'Presser Return...'     *
[illegible]
        move.w    #8,-(a7)               * attend que la touche Return     *
        trap      #1                     * soit pressee...                 *
        addq.l    #2,a7
        cmp.b     #13,d0
        bne       attret
        rts
=
=
[illegible]
[illegible]   dc.b      $1b,$45,13,10,9,9,9
        dc.b      "*************************************",13,10
        dc.b      9,9,9,"*                                   *",13,10
        dc.b      9,9,9,"* ",$1b,$70
        dc.b      " Installation de DISQUE VIRTUEL ",$1B,$71
        dc.b      " *",13,10,9,9,9
        dc.b      "*     (taille minimum = 12 Ko)      *",13,10
        dc.b      9,9,9,"*                                   *",13,10
        dc.b      9,9,9,"*************************************"
        dc.b      13,10,10,10,0
[illegible]   dc.b      9,"Il y a deja trop de disques implantes !",13,10
        dc.b      9,"Le disque virtuel ne peut donc etre installe...",7,0
[illegible]   dc.b      13,10,9
        dc.b      "Allocation memoire pour le disque virtuel impossible !",0
[illegible]   dc.b      13,10,10,9,"Pressez la touche ",$1b,$70," Return ",$1b,$71
        dc.b      "pour retourner au Bureau...",0
[illegible]   dc.b      9,"   0 Ko reserves pour le disque virtuel C:",13,10
        dc.b      9,"Utilisez 'Installer un disque' (option du Bureau)"
        dc.b      13,10,9,"pour l'implantation de ce disque virtuel...",0
[illegible]   dc.b      9,"   0 Ko disponibles pour le disque virtuel",13,10
        dc.b      9,"Combien de Ko voulez-vous reserver ?",13,10
        dc.b      9,"Laissez-en un minimum pour une application !",13,10,10
        dc.b      9,"Pressez ",$1b,$70,"Return",$1b,$71,13,10
        dc.b      9,"pour laisser 64 Ko a l'application ",13,10,9,"? ",0
[illegible]grande dc.b  9,"Votre nombre de Ko est superieur au nombre disponible !"
        dc.b      13,10,7,0
[illegible]petite dc.b  9,"Nombre de Ko insuffisant pour un disque virtuel !"
        dc.b      13,10,9,"Installation non effectuee !",7,0
[illegible]   dc.b      6,0,"        "
[illegible]   dc.b      13,10,10,0
[illegible]
[illegible]
[illegible]   ds.1      2
[illegible]
```

3.3. La ligne « A »

A la conception du microprocesseur 68000, les codes binaires (que nous traduirons sous leur forme condensée hexadécimale) correspondant aux instructions du processeur ont été groupés de sorte qu'aucun de ces codes ne commence par la séquence binaire 1010 (soit « A » en hexadécimal) ou la séquence binaire 1111 (soit « F » en hexadécimal). Rappelons qu'un code d'instruction 68000 occupe toujours deux octets (16 bits), le ou les opérande(s) éventuel(s) occupant deux à huit octets. Toutes les séquences de départ autres que « A » ou « F » sont utilisées et correspondent souvent à un ensemble d'instructions (ainsi la séquence « 6 » correspond à un branchement, la séquence « 9 » à une soustraction et la séquence « D » à une addition, etc.).

Les codes d'instruction 68000 commencant par le quartet « A » ou « F » sont dits réservés mais peuvent être émulés, c'est-à-dire que lorsque le processeur rencontre un code d'instruction commençant par « A » ou « F », il se déroute vers un vecteur d'exception. Ce vecteur est placé à l'adresse $28 pour la ligne « A » et à l'adresse $2C pour la ligne « F ». A ces adresses doivent être placés des adresses de routines prenant en compte ces exceptions. Ces routines peuvent simplement signaler l'exception ou même être constituées d'un « RTE », elles peuvent également, à partir des bits suivant le quartet « A » ou « F » (donc à partir du code complet de l'instruction) dérouter l'exécution vers tel ou tel sous-programme. Dans ce cas les lignes « A » ou/et « F » sont implementées. C'est le cas avec le 520 ST.

La ligne « F », très particulière, fait l'objet du sous-chapitre 3.4. Nous nous intéresserons ici à la ligne « A », implementé par les concepteurs du système d'exploitation dans l'unique but de simplifier le travail des programmeurs en assembleur ! Seize codes opératoires (A000 à A00F) sont disponibles et fournissent une boîte à outils primitive mais efficace, entièrement orientée vers les fonctions graphiques fondamentales. Les routines correspondant aux codes d'instruction A000 à A00F constituent en fait le noyau du VDI et sont fréquemment appelées par les fonctions VDI classiques.

Les instructions de la ligne « A » ont les fonctions suivantes :

A000 Initialisation de la ligne « A »
A001 Fixe la valeur d'un point de l'écran
A002 Demande la valeur d'un point de l'écran
A003 Traçage d'une ligne entre 2 points quelconques
A004 Traçage d'une ligne horizontale
A005 Traçage d'un rectangle rempli
A006 Traçage d'un polygone rempli

A007 Transfert d'un bloc de points
A008 Écriture d'un caractère
A009 Visualisation de la souris
A00A Non-visualisation de la souris
A00B Changement de la forme de la souris
A00C Effacement d'un spirite
A00D Dessin d'un spirite
A00E Copie (avec transformation éventuelle) de bloc de points
A00F Remplissage de zone

Chacune de ces fonctions fait l'objet d'une étude détaillée dans les pages suivantes. Les variables et blocs de variables pointés par les paramètres de retour de A000 ont une importance primordiale pour le bon déroulement des instructions A001 à A00F et doivent être parfaitement compris. Le parallélisme entre les fonctions de la ligne « A » et les fonctions VDI est signalé lorsqu'il existe. Seule la fonction A000 à des paramètres de sortie. Toutes les fonctions, sauf A000 et A00A, ont des paramètres d'entrée (cf. exemples supra pour l'envoi des paramètres).

A000 : Initialisation de la ligne « A »

Aucun paramètre d'appel. Paramètres de retour :

D0.L = pointeur vers le bloc des variables de la ligne « A ».
A0.L = D0.L = pointeur vers le bloc des variables de la ligne « A ».
A1.L = pointeur vers les 3 pointeurs des polices de caractères système.
A2.L = pointeur vers table des 16 adresses de routines de la ligne A.

Le pointeur retourné dans A1 est particulièrement précieux pour l'appel de la fonction VDI Escape 102 (cf. 3.1.). Il peut également servir pour A008. Le pointeur retourné par A2 est indispensable pour une extension de la ligne « A » (étudiée ultérieurement). Enfin le pointeur retourné dans D0 et A0 est DÉCISIF puisqu'il permet l'utilisation des instructions A001 à A00F. L'adresse pointée est celle d'un bloc de variables, propre à l'application ouverte mais utilisée par le VDI, raison pour laquelle l'emploi de fonctions VDI et ligne « A » dans un même programme nécessite, lors de chaque appel de la ligne A, une réinitialisation des variables pouvant avoir été modifiées par le VDI.

Ce bloc comprend 122 octets. Le tableau ci-dessous récapitule les fonctions de chaque variable avec l'accès indexé hexadécimal à la variable et le nom de la variable :

S0.W	plans	nombre de plans vidéo (1 si haute, 2 si moyenne résolution, 4 si basse résolution).

$2.W	octlgn	nombre d'octets par ligne vidéo ($50 si haute résolution, $A0 si moyenne ou basse résolution).
$4.L	CONTRL	pointeur vers la zone CONTRL (cf. VDI, annexe 2).
$8.L	INTIN	pointeur vers la zone INTIN (cf. VDI, annexe 2).
$C.L.	PTSIN	pointeur vers la zone PTSIN (cf. VDI, annexe 2).
$10.L	INTOUT	pointeur vers la zone INTOUT (cf. VDI, annexe 2).
$14.L	PTSOUT	pointeur vers la zone PTSOUT (cf. VDI, annexe 2).
$18.W	vplan0	valeur de la couleur de traçage pour le plan 0.
$1A.W	vplan1	valeur de la couleur de traçage pour le plan 1.
$1C.W	vplan2	valeur de la couleur de traçage pour le plan 2.
$1E.W	vplan3	valeur de la couleur de traçage pour le plan 3.
$20.W	semlgn	drapeau de nombre de lignes à tracer (0 si une ligne, 1 si plusieurs). Sert au système pour joindre correctement les lignes. Inutilisé par la ligne A. Doit être mis à 0.
$22.W	masqlgn	masque pour tracé de ligne (= vsl_style du VDI).
$24.W	modecri	mode d'écriture (étudié en A003).
$26.W	x_orig	coordonnée horizontale du point de départ.
$28.W	y_orig	coordonnée verticale du point de départ.
$2A.W	x_fin	coordonnée horizontale du point de fin.
$2C.W	y_fin	coordonnée verticale du point de fin.
$2E.L	matremp	pointeur vers la matrice de remplissage (v. A004).
$32.W	facrep	facteur répétitif matrice de remplissage (v. A004).
$34.W	draremp	drapeau de remplissage multi-plan (0 si remplissage mono-plan, 1 si remplissage multi-plan).
$36.W	semrest	drapeau de restriction d'affichage (= vs_clip).
$38.W	xminres	valeur horizontale minimum affichage restreint.
$3A.W	yminres	valeur verticale minimum affichage restreint.
$3C.W	xmaxres	valeur horizontale maximum affichage restreint.
$3E.W	ymaxres	valeur verticale maximum affichage restreint.
$40.W	accucad	accumulateur pour cadrage caractère (voir A008).
$42.W	pentcad	pente pour cadrage de caractère (voir A008).
$44.W	direch	direction du cadrage (voir A008).
$46.W	espapol	drapeau de police de caractère (1 si mono-espacée, 0 si non mono-espacée, voir v_justified du VDI).
$48.W	xcarpol	coordonnée horiz. du caractère dans la police.
$4A.W	ycarpol	coordonnée vert. caractère dans la police (cf. infra).
$4C.W	xcarec	coordonnée horizontale du caractère à l'écran.
$4E.W	ycarec	coordonnée verticale du caractère à l'écran. Les coordonnées horizontale et verticale du caractère à l'écran pointent *le coin haut gauche* du caractère et non le coin gauche de la ligne d'écriture du bas (baseline) comme en VDI.

[illegible]	largcar	largeur d'une cellule de caractère en pixels.
[illegible]	hautcar	hauteur d'une cellule de caractère en pixels.
[illegible]	adrpol	pointeur vers l'adresse des *données* de la police de caractères utilisée (voir plus bas).
[illegible]	tailpol	taille de la police de caractère utilisée (cf. infra).
[illegible]	stylaff	vecteur d'effets spéciaux (avec signification des bits identique au paramètre d'appel de vst_effects. *Le soulignement n'est pas pris en compte.*
[illegible]	masqgri	masque pour caractère grisé (bit 1 de vst_effects).
[illegible]	masqita	masque pour caractère italique (bit 2 vst_effects).
[illegible]	largep	largeur en pixels caractère gras (voir plus bas).
[illegible]	itadroi	déplacement à droite d'italique (voir plus bas).
[illegible]	itagau	déplacement à gauche d'italique (voir plus bas).
[illegible]	semech	drapeau d'échelle (0 si pas d'échelle, 1 sinon, c'est-à-dire si la hauteur de caractère à afficher est différente de la hauteur dans la police).
[illegible]	vecrota	vecteur de rotation de caractère (0, 900, 1800 ou 2700, voir vst_rotation du VDI).
[illegible]	coultex	couleur d'affichage du texte.
[illegible]	tampeff	pointe le tampon de calcul d'effets spéciaux.
[illegible]	depeff	déplacement par rapport au pointeur précédent du tampon de calcul d'échelle (voir A008).
[illegible]	coulfon	couleur du fond de caractère lors d'affichage.
[illegible]	typcop	drapeau de type de copie de blocs de bits (0 si type opaque, sinon type transparent, voir VDI, vro_cpyfm et vrt_cpyfm).
[illegible]	adrcremp	pointe une routine exécutée lors du remplissage de zone (voir A00F).

Nombre de ces variables coïncident très précisément à des variables VDI [illegible] des fonctions GEM (Annexe 2) permet d'en appréhender aisément [illegible] l'utilisation. Certaines variables sont par contre propres à la ligne [illegible] ou offrent des possibilités d'emploi supérieures, sans parler des opé[illegible] plus ou moins bizarres qu'elles autorisent. Ces dernières feront donc [illegible] de développements particuliers dans l'étude des fonctions qui les [illegible].

L'étude des en-têtes de polices de caractères dépasserait quelque peu le [illegible] de cet ouvrage. Certaines variables y font cependant référence, aussi [illegible] minimum de connaissances en la matière paraît indispensable.

Une police de caractères peut être séparée en 3 parties : l'en-tête, qui [illegible] les variables descriptives de la police (90 octets) ; la table des déplacements qui comporte autant de mots plus un qu'il y a de caractères dans

la police et qui permet de repérer le point de départ de la matrice d'un caractère donné ; les données de la police, suite de bits correspondant aux matrices de caractères.

Les données des matrices de caractères sont ordonnées *par ligne d'écran* et non par caractère. Ainsi, pour une police 8*16 de 128 caractères, les 128 premiers octets de données correspondront aux 128*8 = 1024 bits des lignes du haut des matrices des 128 caractères. Puis sera codée la deuxième ligne pour chaque caractère, la troisième, etc., jusqu'à la seizième et dernière. Si la matrice de caractère ne faisait que 7 points de large, 128*7 = 896 bits coderaient la première ligne et ainsi de suite.

Si l'on conserve l'une des polices de caractères système lorsque l'on use de la ligne A (polices dont les adresses seront fournies par 0(A1), 4(A1), 8(A1), après appel de A000), les variables nous intéressant dans la police sont les suivantes (déplacement par rapport à l'adresse de départ de la police) :

$36.W	(itagau)	voir vqt_fontinfo du VDI.
$38.W	(itadroi)	voir vqt_fontinfo du VDI.
$3A.W	(largep)	largeur de caractère gras.
$3E.W	(masqgri)	masque pour caractère grisé.
$40.W	(masqita)	masque de rotation de matrice pour italique.
$44.L		adresse de la table des déplacements.
$4C.L	(adrpol)	adresse des données de la police.
$50.W	(tailpol)	taille de la police (nombre d'octets total pour une ligne d'affichage de tous les caractères).
$52.W	(hautcar)	hauteur d'une matrice de caractère de la police.

La largeur et la hauteur de la matrice de caractère peuvent naturellement être connues par la fonction VDI vqt_attributes. La coordonnée horizontale d'un caractère dans la police, xcarpol, peut être calculée à partir de $44.L de l'en-tête de la police (adresse de la table de déplacements), il suffit d'ajouter à cette adresse :

2 * (code ASCII du caractère — premier code ASCII de la police)
Le premier code ASCII de la police est contenu dans $24 de l'en-tête.

A001. Fixe la valeur d'un point de l'écran

Appel :INTIN[0] = valeur du pixel
PTSIN[0] = coordonnée horizontale du pixel.
PTSIN[1] = coordonnée verticale du pixel.

Rappelons que les tableaux CONTRL, INTIN, PTSIN, INTOUT et PTSOUT, sont des *tableaux de mots* (2 octets). Avec un moniteur monochrome, A001 retourne 0 si le pixel est à 0 (blanc), 1 si le pixel est à 1 (noir).

[illegible] 4 couleurs, la valeur du pixel est comprise entre 0 et 3. Avec 16 cou[illegible], elle est comprise entre 0 et $F (voir 1.1.2 pour les couleurs [illegible]ondantes).

[illegible] Demande la valeur d'un point de l'écran

[illegible] : PTSIN[0] = coordonnée horizontale du pixel.
PTSIN[1] = coordonnée verticale du pixel.

Renvoie dans D0.W la valeur du pixel (voir A001).

[illegible]. Traçage de ligne entre deux points quelconques

[illegible] : *indispensables* x_orig = coord. horizontale de départ.
y_orig = coord. verticale de départ.
x_fin = coord. horizontale de fin.
y_fin = coord. verticale de fin.

utiles vplan0 = valeur de la couleur de traçage plan 0.
vplan1 = valeur de la couleur de traçage plan 1.
vplan2 = valeur de la couleur de traçage plan 2.
vplan3 = valeur de la couleur de traçage plan 3.
semlgn = 0 (pour un traçage correct).
masqlgn = masque pour tracé de ligne.
modecri = mode d'écriture.

Toute ligne est toujours tracée de la gauche vers la droite. Le masque de style de ligne est donc appliqué à partir du point gauche. Ce masque subit une rotation vers la gauche égale à : (x_fin − x_orig + 1) modulo 16. Cette rotation est justifiée par la possibilité en VDI de tracer plusieurs lignes successives. Il est donc recommandé de réinitialiser masqlgn avant chaque routine de tracé.

Le mode d'écriture doit *impérativement* être compris entre 0 et 3. Il s'agit du mode défini en VDI par vswr_mode. Les « valeurs de couleur de traçage » des plans 0 à 3 sont égales soit à 0, soit à 1. En monochrome, il suffit de définir la variable vplan0 et en 4 couleurs les variables vplan0 et vplan1. Les valeurs de vplan0 à vplan3 correspondent aux bits 0 à 3 de la couleur définie par vsl_color en VDI (mais *attention*, l'appel de vsl_color ne positionne pas les variables vplan0 à vplan3). Pour un plan donné, tout dépend du mode d'écriture :

— mode REPLACE : si vplan = 0, le pixel de la ligne est mis à 0. Sinon, le pixel de la ligne est mis à 0 ou à 1 selon le masque de ligne (masqlgn).

— mode TRANSPARENT : si vplan = 0, le pixel est inchangé si le masque

est à 0, il est mis à 0 si le masque est à 1. Sinon (vplan différent de 0), le pixel est inchangé si le masque est à 0, il est mis à 1 si le masque est à 1.

— mode XOR : vplan n'intervient pas. Si le masque est à 0, le pixel est inchangé. Si le masque est à 1, la valeur du pixel est inversée.

— mode INVERSE TRANSPARENT : si vplan = 0, le pixel est inchangé si le masque est à 1, il est mis à 0 si le masque est à 0. Sinon, le pixel est inchangé si le masque est à 1, il est mis à 1 si le masque est à 0.

A004. Traçage d'une ligne horizontale

Appel : *indispensables* x_orig = coord. horiz. point de départ.
y_orig = coord. vert. point de départ.
x_fin = coord. horiz. point de fin.
y_fin = coord. vert. point de fin.

utiles vplan0 = valeur de la couleur de traçage plan 0.
vplan1 = valeur de la couleur de traçage plan 1.
vplan2 = valeur de la couleur de traçage plan 2.
vplan3 = valeur de la couleur de traçage plan 3.
modecri = mode d'écriture (cf. vswr_mode du VDI).
matremp = pointeur vers la matrice de remplissage.
facrep = masque de la matrice de remplissage.
draremp = drapeau de remplissage (0 = monoplan).

On pourrait croire que le traçage d'une ligne horizontale ne constitue qu'un cas particulier et simplifié du traçage d'une ligne. Or c'est l'inverse ! Le traçage de ligne horizontale constitue en fait une sous-routine utilisée par les fonctions de remplissage, d'où l'influence de matremp, facrep et draremp.

Les matrices de remplissage, définies par les fonctions VDI vsf_interior, vsf_style et vsf_udpat, appartiennent aux fonctions les plus obscures, et naturellement, les moins utilisées du GEM. Aussi paraît-il utile de rentrer ici un peu dans les détails :

— L'instruction vsf_interior définit un type de remplissage. Avec le paramètre d'appel 2, elle définit le style « matrice » auquel nous nous intéressons ici.

— L'instruction vsf_style, dès lors que le type « matrice » a été choisi, définit un type de matrice (matrice pointillée si style = 4, pleine si style = 8, briques si style = 9, etc.). La matrice système ainsi définie est une matrice de 16 pixels * x pixels (ou x = facrep), recopiée autant de fois que nécessaire. Si l'on travaille avec un moniteur couleur, cette matrice sera sur 2 ou 4 plans selon le mode, autrement dit comportera 2 ou 4 fois (16 * x bits). Les seize premiers bits (sur un ou plusieurs plans) définissent la matrice pour 16 pixels d'une ligne d'écran, les seize suivants pour la ligne d'en-dessous, etc.

— L'instruction vsf_udpat permet de définir sa propre matrice de remplissage, laquelle doit comporter 16 mots pour chaque plan, selon le principe décrit ci-dessus.

L'instruction A004, de même que les instructions A005, A006 et A00F, utilisent la matrice de remplissage. Si celle-ci n'est pas initialisée par le programmeur, la dernière matrice définie par vsf_interior et vsf_style est valide. Si aucun style et aucune matrice n'a été définie même à l'appel de v_opnvwk (INTIN[7] et INTIN[8]), le style est défini à 0 par défaut.

Le drapeau draremp est simplement associé à la résolution mono ou multi-plan de l'écran. Par contre facrep, facteur répétitif de la matrice de remplissage, est une variable système intéressante. Elle définit le nombre de lignes de la matrice, c'est-à-dire le nombre de lignes moins une après lesquelles le motif initial est repris. On a :

facrep = 3 (4 lignes) si vsf_interior = 2 et vsf_style < 8.
facrep = 7 (8 lignes) si vsf_interior = 2 et vsf_style >= 8,
ou vsf_interior = 3 et vsf_style < 6.
facrep = 15 (16 lignes) si vsf_interior = 3 et vsf_style >= 6,
ou vsf_interior = 4 (style utilisateur).

Grâce à la ligne A, il est toutefois possible de modifier le facrep pour une matrice système. Il est surtout possible de fabriquer une matrice de remplissage avec un facteur répétitif quelconque, par exemple de 32 ou 48 lignes de hauteur. Il suffit de placer l'adresse de sa matrice dans matremp, de positionner le drapeau de mono-plan selon la résolution employée et de placer le nombre de lignes −1 de la matrice dans facrep.

Si vous désirez utiliser une matrice de remplissage système conjointement à la ligne « A », il est nécessaire d'utiliser les fonctions VDI vsf_interior et vsf_style, aucune instruction de la ligne « A » ne le permettant. Mais, ATTENTION, la matrice n'est réellement initialisée et disponible pour la ligne « A » que lorsqu'une fonction VDI a été appelée APRÈS les fonctions vsf_interior et vsf_style. Une méthode qui en vaut une autre consiste à appeler deux fois de suite vsf_style... Naturellement, une initialisation de matrice utilisateur, une fois appelé vsf_interior avec le mode 4, peut s'effectuer en assembleur sans passer par vsf_udpat...

A005. Traçage d'un rectangle rempli.

Appel : *indispensables* x_orig = coord. horiz. d'un sommet.
y_orig = coord. vert. d'un sommet.
x_fin = coord. horiz. du sommet opposé.
y_fin = coord. vert. du sommet opposé.

utiles vplan0 = valeur de la couleur pour le plan 0.
vplan1 = valeur de la couleur pour le plan 1.
vplan2 = valeur de la couleur pour le plan 2.
vplan3 = valeur de la couleur pour le plan 3.
modecri = mode d'écriture (cf. vswr_mode).
matemp = pointe la matrice de remplissage.
facrep = facteur de répétition de la matrice.
draremp = drapeau de remplissage mono-plan.
semrest = drapeau de restriction d'affichage.
xminres = minimum horizontal affichage restreint.
yminres = minimum vertical affichage restreint.
xmaxres = maximum horizontal affichage restreint.
ymaxres = maximum vertical affichage restreint.

Les variables de remplissage et de mode d'écriture fonctionnent comme décrit pour A004. Le drapeau semrest correspond au mode de vs_clip. Soit il est à 0 et aucune restriction d'affichage n'existe, soit il est à 1 et le rectangle tracé sera égal à l'intersection du rectangle dont les deux sommets opposés sont fournis et du rectangle restreint défini par xminres, yminres, xmaxres et ymaxres.

A006. Traçage d'une ligne écran d'un polygone rempli.

Appel : *indispensables* CONTRL[1] = nombre de vecteurs.
PTSIN [] = zone des vecteurs.
y_orig = coord. vert. de la ligne.

utiles vplan0 = valeur de la couleur pour le plan 0.
vplan1 = valeur de la couleur pour le plan 1.
vplan2 = valeur de la couleur pour le plan 2.
vplan3 = valeur de la couleur pour le plan 3.
modecri = mode d'écriture (voir vswr_mode).
matremp = pointeur de matrice de remplissage.
facrep = facteur de répétition du remplissage.
draremp = drapeau mono ou multi-plan.
semrest = drapeau d'affichage restreint.
xminres = coord. hor. minimum d'affichage restreint.
yminres = coord. vert. minimum affichage restreint.
xmaxres = coord. hor. maximum affichage restreint.
ymaxres = coord. vert. maximum affichage restreint.

Le polygone à remplir est défini par un certain nombre de vecteurs, c'est-à-dire de coordonnées (x1, y1), (x2, y2), ... (xn, yn), (x1, y1). Le nombre de ces vecteurs est placé dans CONTRL[1]. On notera que ce polygone est fermé, le premier point étant aussi le dernier. L'instruction A006 trace *une*

[illegible] d'écran de ce polygone. Cette ligne peut être composée d'un ou plu[illegible] segments selon les coordonnées des vecteurs, elle peut même être nulle [illegible] coordonnée verticale sort du domaine du polygone. Pour remplir entiè[illegible] polygone, il faut boucler en décrémentant y_orig de l'ordonnée [illegible] du polygone à la plus basse.

[illegible]. Transfert d'un bloc de bits

[illegible] : A6 doit pointer sur un tampon de 76 octets ou :

S0(A6).W = largeur du bloc en points.
S2(A6).W = hauteur du bloc en points.
S4(A6).W = nombre de plans de la destination.
S6(A6).W = index de couleur pour 0 (cf. vrt_cpyfm).
S8(A6).W = index de couleur pour 1 (cf. vrt_cpyfm).
SA(A6).B = mode de transfert (voir plus bas).
SB(A6).B = $C si REPLACE, $D si REVERSE TRANSPARENT, 0 dans les autres modes.
SC(A6).B = $3 si REPLACE, $7 si TRANSPARENT, 0 sinon.
SD(A6).B = $F si REPLACE, 0 sinon.
SE(A6).W = origine horizontale du bloc source.
S10(A6).W = origine verticale du bloc source.
S12(A6).L = pointeur mémoire MFDB source (cf. infra).
S16(A6).W = 2 * nombre de plans de la source.
S18(A6).W = nombre d'octets par ligne d'écran pour source.
S1A(A6).W = 0 si mode vro_cpyfm, 2 si mode vrt_cpyfm.
S1C(A6).W = origine horizontale du bloc destination.
S1E(A6).W = origine verticale du bloc destination.
S20(A6).L = pointeur mémoire MFDB destination (cf. infra).
S24(A6).W = 2 * nombre de plans de la destination.
S26(A6).W = nombre d'octets par ligne d'écran pour dest.
S28(A6).W = 2 dans tous les cas.
S2A(A6).L = matremp (adresse matrice de remplissage) 0 si non prise en compte.
S2E(A6).W = 2 si matremp non nul, 0 sinon.
S30(A6).W = 0 si matrice mono-plan, $20 si multi-plan.
S32(A6).W = facrep, facteur de répétition du motif.
S34(A6) à $4B(A6) doivent être nuls.

[illegible] transfert de bloc de bits copie un bloc de bits d'une source vers une [illegible]ion. Le bloc source et le bloc destination ont la même taille. La copie [illegible] être effectuée en mode vro_cpyfm ou vrt_cpyfm, avec un mode de trans[illegible] logique inférieur à 4 en mode vrt_cpyfm et quelconque en mode [illegible]_cpyfm :

0 place tous les pixels du bloc de destination à 0.
1 résultat = ET logique entre état source et état destination.
2 résultat = ET logique entre état source et inverse état destination.
3 résultat = état de la source (mode REPLACE).
4 résultat = ET logique entre inverse état source et état destination.
5 résultat = état de la destination.
6 résultat = OU exclusif entre source et dest. (mode XOR).
7 résultat = OU logique entre source et destination (TRANSPARENT).
8 résultat = inverse de (source OU destination).
9 résultat = inverse de (source XOR destination).
A résultat = inverse de l'état de la destination.
B résultat = source OU (inverse de la destination).
C résultat = inverse de la source.
D résultat = (inverse de la source) OU destination (INVERSE TRANSPARENT).
E résultat = inverse de (source ET destination).
F place tous les pixels du bloc de destination à 1.

Un bloc de mémoire MFDB est une structure décrivant un ensemble de bits. Cette structure est d'ailleurs implementée dans le compilateur C de Digital Research sous l'appellation FDB. Nous fournirons donc ici sa description en tant que structure C (fichier à inclure : GEMDEFS.H) :

long	fd_addr	adresse de départ du bloc de points.
int	fd_w	largeur en pixels du bloc de points.
int	fd_h	hauteur en pixels du bloc de points.
int	fd_wdwidth	largeur en *mots* du bloc de points.
int	fd_stand	0 si dépend du périphérique, 1 sinon.
int	fd_nplanes	nombre de plans vidéo (1, 2 ou 4).
int	fd_r1	réservé (0 en principe).
int	fd_r2	réservé (0 en principe).
int	fd_r3	réservé (0 en principe).

L'instruction A007 est spécifique de la ligne « A », elle correspond à la primitive de base du remplissage de forme. Son intérêt tient aux possibilités offertes par la manipulation des modes autorisés et surtout par la possible utilisation de matrice de remplissage faisant office de masque. Il est même envisageable, en falsifiant les données sur le nombre de plans, de simuler des effets de dédoublement d'image ou autres effets spéciaux tout à fait irréalisables en C.

A008. Écriture d'un caractère

Appel :
indispensables modecri = mode d'écriture (0 à $13).
xcarpol = coord. horiz. du caractère police.

ycarpol = coord. vert. du caractère police.
xcarec = coord. horiz. du caractère écran.
yecran = coord. vert. du caractère écran.
adrpol = adresse de la police de caractères.
tailpol = taille de la police de caractères.
coultex = couleur du texte affiché.
coulfon = couleur du fond de caractère...

utiles accucad = accumulateur pour cadrage de caractère.
pentcad = pente pour cadrage de caractère.
direch = direction du cadrage de caractère.
largcar = largeur du caractère en pixels.
hautcar = hauteur du caractère en pixels.
stylaff = style d'affichage.
masqgri = masque pour caractère grisé.
masqita = masque pour caractère italique.
largep = taille pour épaississement de caractère.
itadroi = déplacement à droite pour italique.
itagau = déplacement à gauche pour italique.
semech = drapeau d'échelle.
vecrota = vecteur de rotation du caractère.
tampeff = adresse du tampon d'effets spéciaux.
depeff = déplacement du tampon d'échelle.

L'énumération ci-dessus des variables « indispensables » et « utiles » [illegible] effrayer le lecteur. Elle n'a pourtant rien que d'ordinaire si l'on con[illegible] que la définition de ces variables remplace l'exécution d'une bonne [illegible] de fonctions VDI. Il est en effet possible grâce à l'instruction A008 [illegible] de police de caractères, de hauteur, de largeur, de couleur, de [illegible] d'écriture, de coordonnées ou de direction d'affichage pour un seul carac[illegible] il est naturellement possible de ne pas en changer en ne modi[illegible] pas les valeurs correspondantes des variables...

Le cadrage et l'échelle font référence à vst_height et vst_point du VDI. [illegible] modification de la hauteur du caractère par rapport à la hauteur de [illegible] dans la police nécessite un ajustement pour affichage par le système. [illegible] ajustement portant sur la largeur et la hauteur, les variables accucad, [illegible], direch et semech permettent cet ajustement. La variable semech est [illegible] s'il y a ajustement, la variable direch est à 0 s'il y a réduction de taille, [illegible] s'il y a augmentation de taille. L'accumulateur accuad devra être initia[illegible] à $8000 avant un appel de A008 utilisant l'échelle. La variable pentcad [illegible]spond au facteur d'ajustement. S'il y a augmentation de taille :

pentcad = 256 * (taille voulue - taille actuelle)/taille actuelle. S'il y a [illegible]tion de taille :

pentcad = 256 * taille voulue / taille actuelle.

A009. Visualisation de la souris

Appel : INTIN [0] : Force la visualisation si nul.

Fonction strictement identique à v_show_c en VDI.

Remarque : Toute demande de visualisation ou de non-visualisation de la souris provoque une incrémentation ou une décrémentation d'un compteur. Si ce compteur est nul, une demande de visualisation est valide et le compteur passe à 1. S'il n'est pas nul, la demande de visualisation n'est pas prise en compte mais le compteur est incrémenté. Par contre, une demande de non-visualisation est toujours prise en compte et provoque l'effacement de la souris avec décrémentation du compteur. Le paramètre d'appel INTIN[0] permet de réinitialiser ce compteur.

A00A. Non-visualisation de la souris

Aucun paramètre d'appel ou de sortie. Comme expliqué ci-dessus, une demande de non-visualisation, alors que la souris est déjà invisible, provoque une décrémentation du compteur, de sorte que la demande de visualisation suivante ne sera pas prise en compte sauf si INTIN[0] = 0.

A00B. Changement de la forme de la souris

Appel : INTIN[0] = coord. horiz. sommet gauche de la forme.
INTIN[1] = coord. vert. sommet gauche de la forme.
INTIN[2] = 1.
INTIN[3] = couleur du masque (normalement 0).
INTIN[4] = couleur des données (normalement 1).
INTIN[5] à INTIN[20] = masque de la forme (16 mots).
INTIN[21] à INTIN[36] = données de la forme (16 mots).

Instruction identique à vsc_form en VDI. La notion de données et de masque d'une forme tient à la nécessité de visualiser la souris quelque soit le fond de l'écran. Ainsi pour la forme de souris flèche, les données correspondent à la flèche (bits mis à la couleur de INTIN[3]) et le masque correspond à un cadre d'un pixel de large auréolant la flèche (bits mis à la couleur de INTIN[4]). Ainsi une flèche noire est-elle visible sur fond noir à cause de son auréole blanche (son « masque »). Seuls les bits à 1 dans les données et le masque sont pris en compte lors de l'affichage.

Comme vsc_form, l'instruction A00B change la forme mais ne provoque pas une visualisation de la souris si celle-ci est cachée.

A00C. Effacement d'un spirite

Appel : A2 = pointe vers un tampon de sauvegarde de 10 octets + 64 octets par plan vidéo minimum.

[illegible] ci-dessous pour l'analyse du « spiritisme ».

[illegible]. Dessin d'un spirite

[illegible] : *D0.W* = coord. horiz. sommet gauche du spirite à tracer.
D1.W = coord. vert. sommet gauche du spirite à tracer.
A0.L = pointeur vers le bloc de définition du spirite.
A2.L = pointeur vers le tampon de sauvegarde du spirite.

Le bloc de définition pointé par A0 à la structure suivante :

0.W déplacement horizontal soustrait à la coord. avant affichage.
2.W déplacement vertical soustrait à la coord. avant affichage.
4.W type d'affichage (positif ou négatif)

coul. données	coul. masque	positif	négatif
0	0	sans effet	sans effet
0	1	coul. masque	coul. masque
1	0	coul. données	xor avec écran
1	1	coul. données	coul. données

6.W couleur du masque
8.W couleur des données
10.W à 5F 32 mots définissant l'image entrelacée du spirite :
mot 0 = ligne 0 du masque, mot 1 = ligne 0 des données,
mot 2 = ligne 1 du masque, mot 3 = ligne 1 des données, ...)

Les routines de dessin et d'effacement de spirite constituent en fait des [illegible]es générales, utilisées par le GEM pour redessiner le pointeur de souris [illegible] l'on déplace celle-ci. Lors d'un déplacement de la souris dépassant [illegible] seuil (voir 5.2), le circuit clavier génère une interruption, laquelle [illegible]que l'exécution d'une routine qui stocke les nouvelles coordonnées de [illegible] et met un drapeau à 1. A chaque interruption balayage vidéo (70 [illegible] par seconde en monochrome), ce drapeau est testé et s'il est placé, une [illegible] est exécutée, laquelle recupère dans le bloc de sauvegarde (pointe par [illegible] dans A00C et A00D) le contenu bit à bit de la zone recouverte par la [illegible], le replace sur l'écran, sauvegarde la partie de l'écran qui va être recou[illegible] par la souris et, enfin, affiche la souris. La primitive A00C est consti[illegible] de l'affichage de l'ancienne zone sauvegardée. La primitive A00D est [illegible]uée de la sauvegarde de la zone qui va être recouverte et de l'affichage [illegible] spirite.

Ces routines permettent de gérer l'affichage et donc le déplacement de [illegible] à une myriade de spirites, surtout si l'on utilise la routine d'interruption [illegible]yage pour générer cet affichage (cf. chapitre 5).

A00E. Copie d'une forme FDB.

Appel : CONTRL[7-8] = adresse du bloc source FDB.
CONTRL[9-10] = adresse du bloc destination FDB.
INTIN[0] = mode d'écriture.
INTIN[1] = couleur affectée aux bits à 1.
INTIN[2] = couleur affectée aux bits à 0.
PTSIN[0] = coord. horiz. du sommet de la source.
PTSIN[1] = coord. vert. du sommet de la source.
PTSIN[2] = coord. horiz. du sommet opposé de la source.
PTSIN[3] = coord. vert. du sommet opposé de la source.
PTSIN[4] = coord. horiz. du sommet de la destination.
PTSIN[5] = coord. vert. du sommet de la destination
PTSIN[6] = coord. horiz. du sommet opposé de la dest.
PTSIN[7] = coord. vert. du sommet opposé de la dest.
typcop = 0 si mode vro_cpyfm, -1 si mode vrt_cpyfm.

INTIN[1] et INTIN[2] ne doivent être définis que lorsque l'on veut effectuer une copie similaire à vrt_cpyfm. Cette instruction constitue en effet la primitive commune à vro_cpyfm et vrt _cpyfm, elle permet de transférer un bloc de bits avec modification éventuelle de ce bloc (selon le mode d'écriture) et de sa taille.

Comme pour l'instruction A007, il est possible de jouer sur les matrices de remplissage et sur le drapeau de mode d'écriture pour obtenir des effets peu ordinaires à l'affichage.

A00F. Remplissage de zone.

Rappel : INTIN[0] = couleur de contour.
PTSIN[0] = coord. horiz. point de départ du remplissage.
PTSIN[1] = coord. vert. point de départ du remplissage.
adrcremp = adresse de routine d'arrêt de remplissage.

L'instruction A00F correspond à la primitive de v_contourfill en VDI. Si la couleur de contour est négative, l'algorithme de remplissage sélectionne comme couleur de contour toute couleur différente de la couleur du pixel de départ. Sinon il n'y a remplissage que si la couleur de contour est différente de la couleur du pixel de départ. Rappelons que la fonction v_contourfill utilise les attributs habituels du remplissage (matrice éventuelle, couleur, etc.).

L'instruction A00F ajoute une possibilité supplémentaire à v_contourfill. On peut, par adrcremp, stopper le remplissage après une ligne donnée. Après chaque traçage de ligne, la routine A00F exécute en effet le sous-programme d'adresse contenue dans adrcremp. Soit cette routine renvoie 0 dans D0 (cas de la routine système) et le remplissage se poursuit, soit elle renvoie une valeur non nulle dans D0 et le remplissage est stoppé.

Tout programmeur peut donc, en utilisant l'instruction A00F, stopper le remplissage, par exemple lorsqu'une certaine hauteur est atteinte, lorsqu'un certain temps est dépassé, lorsqu'une touche est appuyée. *ATTENTION*, dans tous les cas adrcremp doit pointer sur une routine, qui sera au minimum :

```
moveq.1 # 0,d0
rts
```

Exemple d'utilisation de la ligne « A »

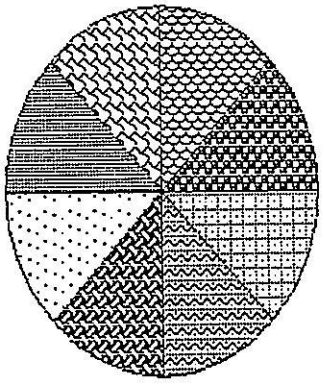
Le programme ci-dessous utilise plusieurs instructions de la ligne « A », à savoir A000, A003, A009, A00A et A00F. Tout programmeur confirmé le trouvera d'une simplicité limpide, quelques précisions paraissent cependant utiles même si le listing est largement commenté :

— Ce programme n'a qu'une fonction : effacer l'écran et afficher un « camembert » partagé en huit secteurs de même taille. Ces secteurs sont remplis avec des matrices différentes. Une attente d'entrée clavier suit l'affichage, avant retour au Bureau GEM.
— Pour effectuer cette tâche, le programme utilise des fonctions VDI, un Trap système et des instructions de la ligne « A »... Abondance de biens ne nuit pas ! Cette courte application est donc une application GEM, contrairement au programme de disque virtuel présenté en 3.2.
— Le lecteur attentif notera que l'utilisation d'une instruction de la ligne « A » comporte une initialisation réduite au minimum. Nombre de paramètres de la ligne « A » n'ont en effet pas besoin d'être réinitialisés, à partir du moment où l'on est certain de leur état, ce qui suppose d'avoir bien assimilé le mode d'emploi de la ligne « A » et du VDI et les résultats d'exécution.
— Pour les utilisateurs du compilateur Digital Research, voici les commandes nécessaires pour assembler et exécuter l'édition de liens (le programme source étant appelé « camembert.s » et placé sur le lecteur B) : as68 -f b: -u -1 b: camembert.s
link68 [tem[b:],u,s] b:camembert.68k = gemstart,b:camembert, vdibind, gemlib
relmod b:camembert

Le fichier objet produit s'appellera naturellement « camembert.prg » et sera exécutable par double cliquage. On relèvera que seul vdibind est lié au programme, aucun appel de fonction AES n'étant effectué par l'application, qui n'a pas besoin non plus de libf, librairie mathématique.

```
.globl   _main                  * declaration obligatoire de main  *
.globl   _contrl                * et des variables de base du VDI  *
.globl   _intin                 * pour une edition de liens cor-   *
.globl   _ptsin                 * recte avec le programme gemstart *
.globl   _intout                * et la bibliotheque GEM et VDI.   *
.globl   _ptsout
.text
_main
         lea       par_ent,a0          * pointe la zone des parametres *
         movea.l   a0,a1               * d'entree pour v_opnvwk. dans  *
         moveq     #11,d0              * laquelle sont stockes douze 1 *
ppar_ent
         move.w    #1,(a0)+
         dbra      d0,ppar_ent
         moveq     #2,d0               * remplace le style de remplis- *
         move.w    d0,14(a1)           * sage interieur par 2. soit le *
         move.w    d0,20(a1)           * style matrice et definit le   *
         move.l    #par_sort.-(a7)     * systeme de coordonnees (RC)   *
         move.l    #identif.-(a7)
         move.l    #par_ent.-(a7)
         jsr       _v_opnvwk           * initialise l'application      *
         adda.l    #12,a7
         move.w    identif.-(a7)       * identificateur de l'application  *
         jsr       _v_clrwk            * effacement de l'ecran            *
         addq.l    #2,a7
         move.w    #3600,-(a7)         * angle de fin : 360 degres        *
         move.w    #0.-(a7)            * angle de depart : 0 degre        *
         move.w    #100,-(a7)          * rayon de l'arc : 100 pixels      *
         move.w    #200.-(a7)          * ordonnee du centre : 200         *
         move.w    #320,-(a7)          * abcisse du centre : 320          *
         move.w    identif.-(a7)
         jsr       _v_arc              * tracage de l'arc                 *
         adda.l    #12,a7
         dc.w      $A000               * APPEL de la LIGNE "A"            *
         move.l    d0,varligna         * stocke adresse bloc variables    *
         movea.l   4(a0),a3            * a3 = adresse de CONTRL ligne "A" *
         movea.l   8(a0),a4            * a4 = adresse de INTIN ligne "A"  *
         movea.l   $c(a0),a5           * a5 = adresse de PTSIN ligne "A"  *
         move.l    #rien,-(a7)         * "rien" parce qu'une adresse est  *
         move.l    #rien,-(a7)         * necessaire ...                   *
         move.l    #rien.-(a7)
         move.w    #320,-(a7)          * ordonnee puis abcisse de la *
         move.w    #540.-(a7)          * souris definies en 320.540  *
         move.w    identif.-(a7)
         jsr       _vrq_locator
         adda.l    #18,a7
         movea.l   varligna,a0         * adresse du bloc var. ligne "A"   *
         move.w    #320.$26(a0)        * x_orig = 320                     *
         move.w    #200.$28(a0)        * y_orig = 200                     *
         move.l    #retnorm.$76(a0)    * stocke une fois pour toutes l'a- *
         lea       speccoor.a1         * dresse de retnorm pour rempliss. *
         moveq     #7.d7               * compteur de boucle               *
bouctra
         move.w    (a1)+.$2a(a0)       * stocke x_fin de la ligne   *
         move.w    (a1)+.$2c(a0)       * stocke y_fin de la ligne   *
         movem.l   d7/a0-a1.-(a7)      * pour sauver les pointeurs  *
         dc.w      $A003               * TRACAGE DE LIGNE           *
         movem.l   (a7)+.d7/a0-a1      * restitue les pointeurs     *
         dbra      d7.bouctra          * etc. pour les 8 lignes     *
         moveq     #7,d7               * nouveau compteur de boucle *
         move.l    a1.-(a7)            * sauve le pointeur de param.*
```

```
boucfon
        moveq     #0,d6                 * compteur de boucle !        *
        move.w    d7,-(a7)              * sauve premier compteur      *
repstyle
        move.w    (a7),-(a7)            * empile compteur de remplissage  *
        addi.w    #14,(a7)              * pour definir le style (21 a 14) *
        move.w    identif,-(a7)         * identificateur d'application    *
        jsr       _vsf_style            * definit le style de remplissage *
        addq.l    #4,a7                 * et recommence une autre fois... *
        dbra      d6,repstyle           * pour prise en compte            *
        move.w    #1,(a4)               * INTIN[0] = 1, couleur du cadre  *
        movea.l   2(a7),a1              * recupere le pointeur de param.  *
        move.w    (a1)+,(a5)            * stocke abcisse puis ordonnee du *
        move.w    (a1)+,2(a5)           * point de depart du remplissage  *
        move.l    a1,2(a7)              * mise a jour ptr de parametres   *
        dc.w      $A00F                 * REMPLISSAGE DE FORME            *
        move.w    (a7)+,d7              * restaure compteur de boucles    *
        dbra      d7,boucfon            * recommence pour les 8 secteurs  *
        movea.l   (a7)+,a1              * pour mise a jour de la pile     *
        move.w    #0,(a4)               * INTIN[0]=0, reset souris        *
        dc.w      $A009                 * VISUALISATION DE LA SOURIS      *
        move.w    #2,-(a7)              * peripherique = clavier          *
        move.w    #2,-(a7)              * fonction = Bconin()             *
        trap      #13                   * attente d'entree clavier        *
        addq.l    #4,a7                 * restitue la pile                *
        dc.w      $A00A                 * NON-VISUALISATION DE LA SOURIS  *
        move.w    identif,-(a7)         * identificateur de l'application *
        jsr       _v_clsvwk             * sort de l'application           *
        addq.l    #2,a7                 * mise a jour de la pile          *
        rts                             * et retour au Bureau...          *
retnorm
        moveq     #0,d0                 * routine executee a chaque rem-  *
        rts                             * plissage de nouvelle ligne      *
.data
par_ent         ds.w        12          * parametres d'entree v_opnvwk    *
par_sort        ds.w        57          * parametres de sortie v_opnvwk   *
identif         ds.w        1           * pour stocker l'identificateur   *
tem             ds.l        1           * pour stocker n'importe quoi !   *
adrligna        ds.l        1           * pour stocker adresse bloc var.  *
_contrl         ds.w        12          * zones de stockage obligatoires  *
_intin          ds.w        128         * pour utilisation du GEM, puis   *
_ptsin          ds.w        256         * coordonnees des points de fin   *
_intout         ds.w        128         * de ligne et de remplissage      *
_ptsout         ds.w        12
spaccoor        dc.w        220,200,250,130,320,100,390,130
                dc.w        420,200,390,270,320,300,250,270
                dc.w        270,190,270,130,370,130,370,190
                dc.w    370,210,370,270,270,270,270,210
.end
```



3.4. Abuser de la ligne « A », user de la ligne « F »

Il est tout à fait envisageable de détourner la ligne « A » à son seul profit, en modifiant l'adresse placée en $28.L, adresse pointant vers la routine système de tri de la ligne « A », et en y placant l'adresse d'une routine « maison » qui pratique le tri entre vos propres instructions « A »... Mais il semble plus intéressant de détourner seulement cette ligne, conservant ainsi seize instructions (A000 à A00F) dont le bon fonctionnement est du moins garanti ! Pour pratiquer ce détournement, il faut écrire une routine de tri semblable à celui-ci :

```
inter"a"
        movea.l $2(a7),a0          * a0 = PC lors de l'interruption  *
        move.w  (a0),d0            * d0 = code de l'instruction "A"  *
        and.w   #$fff,d0           * supprime le quartet "A" commun  *
        addq.l  #2,2(a7)           * ainsi PC pointe après instruct. *
        cmpi.w  #$f,d0             * Si code instruction > A00F,     *
        bhi     personnel          * aller au tri personnel          *
        movea.l adtable,a0         * a0 pointe la table des routines *
        bra     commun             * système ligne "A", retournee    *
personnel                          * dans A2 à l'appel de A000       *
        sub.w   #$10,d0            * si code > A00F , le compare au  *
        cmp.w   d0,maxiligna       * code maxi stocke par vous dans  *
        bhi     retour             * "maxiligna". Si >, fin          *
        movea.l adtpers,a0         * sinon a0 pointe la table de vos *
commun                             * adresses de routines            *
        lsl.w   #2,d0              * x 4 pour déplacement dans table *
        movea.l 0(a0,d0),a1        * a1 = adresse dans la table      *
        movem.l d3-d7/a3-a5,-(a7)  * pour garantir leur survie...    *
        jsr     (a1)               * execute la routine ligne "A"    *
        movem.l (a7)+,d3-d7/a3-a5  * pour les ressusciter...         *
retour
        rte                        * ATTENTION : rte !            *
```

L'adresse de cette routine sera placée en $28.L, ce placement étant effectué en mode superviseur, *après* avoir initialisé la ligne A par A000, initialisation fournissant le pointeur A2 de la table système. Ce pointeur sera stocké dans la variable « adtable ». La variable « maxiligna » devra naturellement avoir été initialisée et « adtpers » devra pointer sur votre table d'adresses de routines.

Il est ainsi possible d'implémenter jusqu'à 4080 nouvelles instructions de la ligne « A » (allant de A010 à AFFF), toutes appellables sur deux octets. Il ne faudra toutefois pas oublier que le niveau de priorité de la ligne « A » est supérieur à celui des interruptions auto-vectoriées, par exemple l'interruption de balayage vertical...

Jusqu'en novembre 1985, les systèmes d'exploitation diffusés pour le 520 ST ignoraient la ligne « F »... Mais la nécessité de réduire le volume du système s'est faite plus grande, l'implantation en ROM ne pouvant être éter-

[illegible]ment reportée ! Et le « truc » bien connu des développeurs sur 68000, [illegible]loi des instructions réservées, en l'occurrence la ligne F puisque le ligne [illegible] déjà partiellement implementée, est devenu inéluctable.

Fin janvier 1986, la ligne « F » demeure exclusivement utilisée dans [illegible], lequel couvre tout de même la moitié du système, mais qui peut pré[illegible] jusqu'où elle débordera lorsque la ROM sera enfin implantée ! Aussi [illegible]formations en ce domaine doivent-elles être prises avec prudence, leur [illegible] de vie pouvant même être inférieure au délai de parution de l'ouvrage [illegible] vous lisez !

Pourtant, il est certain que la ligne « F » demeurera implantée sur le [illegible] ST, même si cette implantation constitue un pis aller puisqu'elle va à [illegible]contre d'une portabilité ultérieure sur la ligne 68000. Rappelons en effet [illegible] la ligne « F » ne constitue pas une ligne « réservée » sur le 68020 et le [illegible] mais bel et bien un index d'instructions co-processeur pour virgule flot[illegible]... Aussi l'information ne saurait-elle demeurer du domaine confiden[illegible], d'autant qu'elle autorise au programmeur un accès rapide et sûr vers [illegible]ques 626 routines, à l'heure actuelle.

Ces routines couvrent les domaines les plus variés, du traitement de chaîne à l'affichage d'icônes en passant par la manipulation de fichiers, le traçage [illegible] fenêtres, la gestion de timer et le déroulement de menus. Mais, plutôt que [illegible] vous mettre simplement l'eau à la bouche, et compte tenu des possibles [illegible]olutions de la ligne « F », il semble préférable d'exposer ici la méthode [illegible]exploitation et d'utilisation de cette ligne :

1) lire le contenu de l'adresse $2C.L. Cette adresse correspond au vecteur [illegible]obal de traitement de la ligne « F ».
2) désassembler la routine débutant à cette adresse. Cette routine sera en [illegible]incipe structurée ainsi :
— dépilement du registre d'état (c'est-à-dire d'un mot),
— test du contenu de l'adresse empilée suivante (qui n'est autre que le compteur de programme) et spécialement de son bit 0.
— empilement du registre d'état, ET logique avec les trois quartets bas du code d'instruction pointé, si bit 0 à 0. Puis, saisie de l'adresse de départ de la table des adresses de routines, ajout du code de l'instruction F (sans le quartet F) et saut absolu à la routine correspondante.
— si le bit 0 du code d'instruction est impair, ET logique entre ce code et SFFF, afin de déterminer les registres à dépiler et fin.
3) A partir de l'adresse de table obtenue par la routine précédente, on disposera des adresses de toutes les routines de la ligne « F » et un désassemblage ou/et des tests permettront de déterminer la fonction, donc l'utilisation de ces routines.

Pour les instructions de la ligne « F » de code impair, la routine se livre à une manipulation des codes du *programme lui-même* qui donne à penser que cette routine devra être modifiée si elle est placée en ROM ou sera plus probablement implantée en RAM.

A titre d'information, mais sans aucune garantie de fonctionnement sur votre système d'exploitation et bien sûr sans garantie non plus de portabilité sur les systèmes ultérieurs, voici quelques fonctions d'instructions de la ligne « F » :

F034 Traçage de rectangle à partir d'une GRECT (voir F050).
F044 Traçage d'une ligne brisée pointillée.
F050 Conversion d'une GRECT (structure C, voir chapitre 2) en ensemble de cinq points pour traçage de rectangle.
F080 Renvoie le minimum de deux nombres.
F128 Renvoie le maximum de deux nombres.
F1CC Concaténation de chaînes.
F1E0 Comparaison de deux chaînes (retourne 1 si égales, 0 sinon).
F250 Calcul de la longueur d'une chaîne.
F268 Compare deux chaînes (retourne 0 si égales, différence sinon).
F26C Recherche d'un caractère dans une chaîne.
F3E4 Conversion d'un caractère en majuscule.
F5D8 Enregistrement d'événements (primitive d'appl_record).
F754 Traçage d'un rectangle en pointillé et avec ombre d'une GRECT.
F758 Traçage d'un rectangle en pointillé à partir d'une GRECT.
F77C Réinitialisation des paramètres de base du VDI.
F898 Copie d'une chaîne.
F9B0 Affichage du contenu d'un fichier à l'écran.

La liste exhaustive de l'ensemble des fonctions de la ligne « F » déborderait le cadre de ce libre (elle l'engloutirait même) et serait de peu d'intérêt dans la mesure où les implantations des routines sont sujettes à évolution tant que le système n'est pas en ROM. D'une façon générale, les routines de l'AES fonctionnant par transfert de paramètres empilés, les instructions de la ligne « F » devraient toutefois se révéler aussi souples d'emploi que les Traps. Il s'agit donc de la bibliothèque de fonctions la plus vaste et peut-être la plus utilisable de tout le logiciel système...

4. LIENS ENTRE LANGAGES

4.1. Compilation et édition de liens

Même aux développeurs professionnels les mécanismes de la compilation et de l'édition de liens restent parfois obscurs. Il est vrai que seule une analyse comparée détaillée des fichiers objets produits à chaque phase du processus de compilation peut permettre d'appréhender ces mécanismes. Pour être honnête, il faut également reconnaître que de notables différences existent entre les divers compilateurs, l'analyse suivante étant essentiellement effectuée à partir du compilateur de Digital Research.

Tout programme ne pouvant être compris d'un microprocesseur que s'il est écrit en langage binaire, les processus de compilation, d'assemblage et d'édition de liens sont destinés à effectuer la conversion d'un langage de programmation en binaire. Pour un programme écrit, par exemple, en langage C, la compilation se déroule en plusieurs phases, chacune de ces étapes consistant en la conversion d'un fichier dans un autre à l'aide des programmes suivants :

— analyseur lexical (CP68). Sa première tâche est de supprimer tous les blancs et commentaires du programme source. Puis il établit une table des valeurs de variables indéfinies, charge chacun des fichiers à inclure (# include), compare les valeurs indéfinies aux « # define » de ces fichiers et opère les conversion nécessaires. De plus il stocke en tête du programme les énoncés de structures de ces fichiers « # include ». Enfin il remplace chacune des valeurs de variables définies dans le programme par un « # define » (ex : « # define flèche 0 » conduira au remplacement de tous les symboles « flèche » par la valeur 0). Le fichier produit a l'appendice « .I ».

— analyseur syntaxique (C068). Il effectue la phase principale de la compilation. Il construit une table des symboles contenant les noms de fonctions, les noms de variables et les étiquettes internes de branchement. Le fichier « .I » précédent est transformé en un fichier à suffixe « .1 », précodé. Ce pré-codage générique n'a pas de rapport avec les mnémoniques 68000, il est semblable au pré-codage effectué pour une conversion en 8086 par exemple. Le fichier produit n'est rien d'autre qu'un arbre syntaxique avec table des symboles et tables de précédence.
— générateur de code (C168). A partir de l'arbre syntaxique (fichier « .1 »), ce programme génère un fichier en langage d'assemblage, c'est-à-dire en mnémoniques 68000. Les numéros de lignes sont ajoutés en commentaires. Les registres de données D0 à D7 sont codés R1 à R8 et les registres d'adresse sont codés R9 à R15. Les branchements locaux (internes aux procédures) sont codés à partir de L1. Le fichier produit a le suffixe « .S ».

A tous les stades de la compilation, les fichiers générés sont en ASCII. Le fichier pré-codé produit par l'analyseur syntaxique n'en est pas pour autant très compréhensible. Sa lecture suppose une connaissance des procédures d'analyse « grammaticale » du langage, de traitement des tables d'analyse ascendante, concepts qui nous éloignent notablement du sujet principal de ce chapitre.

Le fichier « .S », produit final de la compilation, est tout à fait semblable à un programme écrit en langage assembleur même si le codage des registres et des labels locaux le rend un peu étrange. Les phases ultérieures d'assemblage et d'édition de liens sont par conséquent identiques selon que l'on désire rendre exécutable un fichier « C » compilé ou un fichier écrit en langage d'assemblage. Ces phases sont :

— L'assemblage (AS68). Conversion du fichier source ASCII en fichier binaire à suffixe « .0 ». Les appels de routines et les noms de variables externes à la procédure assemblée sont traduits par « $0 », qui indique une référence aux deux tables des noms de routines et des noms de variables placées à la suite du fichier objet. La table des noms de routines est elle-même séparée en deux : table des labels externes et table des labels internes. Le fichier objet produit débute par une instruction de branchement suivie des caractéristiques du programme selon le schéma :

  mot long : nombre d'octets du texte (programme)
  mot long : nombre d'octets de la zone des données
  mot long : nombre d'octets réservés pour la pile
  mot long : nombre d'octets de la table des symboles
  puis 10 octets nuls (réservés).
— L'édition de liens (LINK68). Un programme écrit en C ou en langage d'assemblage peut faire appel à des fonctions externes, par exemple des

fonctions GEM. L'édition de liens est destinée à rechercher dans des bibliothèques les routines correspondant à ces appels et à associer au fichier objet ces routines. De plus, à tout programme devant fonctionner dans l'environnement GEM doit être associé le fichier « GEMSTART » d'initialisation des paramètres. Les bibliothèques essentielles sont :

OSBIND : fonctions système (Traps, voir Annexe 4)
VDIBIND : fonctions VDI (voir Annexe 2)
AESBIND : fonctions AES (voir Annexe 2)
GEMLIB : instructions C (comme printf, voir Annexe 1)
LIBF : fonctions mathématiques.

Le fichier produit par l'édition de liens comporte le suffixe « .68K » et possède le même bloc de départ que le fichier « .0 » (voir ci-dessus). Il faut toutefois noter que l'édition de liens gonfle sérieusement le volume du fichier objet, du fait de l'adjonction des routines externes et de la « table des symboles de liens » laquelle contient les détails sur les symboles globaux utilisés. Les sauts à des routines sont codés « jmp $xy » (ou « jsr $xy »), où xy correspond à un déplacement relatif dans la table des symboles.

— Le générateur de code exécutable (RELMOD). Le fichier « .68k » produit par l'édition de liens doit enfin être converti en fichier exécutable, à suffixe « .PRG » par le programme RELMOD. En effet le fichier produit par l'édition de liens contient 4 parties :

Le code objet du programme
La zone réservée aux données initialisées du programme,
La table des symboles avec le déplacement relatif au début du programme et le type (variable, label),
Une table de précédence indiquant le type de chaque *mot* du programme, les adresses relogeables étant codées 0000.

Le programme RELMOD laisse inchangés les trois premières parties mais convertit la dernière table en une table d'octets : en ajoutant le premier octet à l'adresse du début de programme on obtient la première adresse relogeable à convertir en adresse absolue, puis en ajoutant à cette adresse le deuxième octet, on obtient la seconde adresse à convertir, etc. La table ainsi obtenue est de taille très réduite par rapport à la première et permet, lors du chargement du programme, une conversion très rapide des adresses relatives en adresses absolues.

Ce survol rapide des diverses phases de la compilation et de l'édition de liens permet de relever que les trois dernières étapes sont indispensables même pour un programme en assembleur puisqu'elles conditionnent la réussite du processus de lancement du programme.

Certains liens existent entre les bibliothèques qui méritent d'être signalés car il occasionnent de fréquentes erreurs :

— la bibliothèque « GEMSTART » (ou son équivalent « ACCSTART » pour les accessoires) est indissociable de la bibliothèque « GEMLIB » (cette dernière implémentant la fonction « _main » pour les initiés).
— la bibliothèque «GEMLIB » nécessite en principe d'être associée à « LIBF », certaines fonctions C utilisant les fonctions de conversion de type. Si toutefois votre programme ne comporte que des fonctions C non mathématiques, les manques signalés à l'édition de liens (ex. « undefined : _etoa, _ftoa ») sont sans importance et ne gênent pas le fonctionnement du programme.

Si l'édition de liens et la génération de code exécutable doivent obligatoirement être pratiquées, rien n'oblige par contre à lier le programme à des bibliothèques, à partir du moment où aucun appel externe n'est effectué et si l'on désire simplement produire un programme TOS, donc n'utilisant pas l'environnement GEM. Un exemple en est d'ailleurs fourni au chapitre 3 avec le programme d'installation d'un disque virtuel. Notons également que le suffixe « TOS » peut remplacer le suffixe « PRG », selon le bon vouloir du programmeur ou de l'utilisateur. Ce suffixe n'est pas texté pour l'exécution, seule la structure du programme et l'appel ou non à des fonctions GEM font qu'un programme tournera sous GEM ou non. Même l'installation d'une application à partir du Bureau (option du Menu) n'a qu'un effet réduit sur l'exécution : on dispose de la souris et d'une barre portant le titre du programme en haut de l'écran si le programme est installé en GEM, on dispose d'un écran blanc si le programme est installé en TOS.

Lors de l'appel d'un programme à partir du Bureau (ou à partir du « COMMAND.PRG » fourni avec le CP/M-68k), le programme est chargé en mémoire à partir de la plus basse adresse disponible. 256 octets sont réservés au début de cette zone pour les informations sur le programme, informations provenant du bloc de départ du programme et adaptées à l'environnement. Ce bloc a la structure suivante :

$0.L. adresse absolue de départ de ce bloc (!)
$4.L adresse du haut de la mémoire vive disponible + 1 (non compris les 32 Ko réservés à la mémoire d'écran)
$8.L adresse absolue de départ du programme (codes)
$C.L. longueur du programme (codes, en nombre d'octets)
$10.L adresse absolue de départ des données initialisées
$14.L longueur de la zone des données initialisées
$18.L adresse absolue de départ des données non initialisées
$1C.L longueur de la zone des données non initialisées

[illegible] adresse de la ligne de commandes du programme
[illegible] adresse de l'environnement (ex : « PATH = .A: \ »)
[illegible] début de la ligne de commandes (si elle existe)

A la suite de ce bloc sont placées les trois zones du programme : codes, données initialisées et données non initialisées. Au chargement, toute la mémoire disponible est réservée et la pile du programme est donc placée juste au-dessous de la mémoire d'écran. Une fois les adresses « relogeables » converties en adresses absolues et après exécution du début du programme (placé par GEMSTART à l'édition de liens), seuls 1024 octets sont réservés à la pile du programme pour la pile, le reste de la mémoire devenant disponible pour les allocations ou l'éventuel chargement d'autres programmes.

Au-delà des mécanismes généraux de la compilation et de la mise en route de programmes décrits ci-dessus, il semble utile d'aborder les principaux problèmes pratiques pouvant survenir :

1. Les erreurs syntaxiques ou les oublis constituent l'immense majorité des erreurs relevées à la compilation. Les erreurs se produisant à l'édition de liens viennent plutôt d'une méconnaissance du contenu des bibliothèques ou de leurs interactions. Nous ne saurions trop conseiller de désassembler les fichiers des bibliothèques pour mieux connaître celles-ci. Une lecture attentive des fichiers de définition leurs correspondant est encore plus nécessaire... et plus aisée.

2. Le compilateur et l'éditeur de liens fonctionnent bien, du moins pour ce qui est du système de Digital Research, et, d'une façon générale, mieux vaut imputer les erreurs survenant à votre programme qu'au compilateur... même si l'erreur signalée semble correspondre à une erreur système. Un exemple classique illustre bien cette règle : l'apparition du message d'erreur « it read error » ou « intermediate read error on », lors de l'assemblage. Cette erreur est uniquement due au fait que votre disque est plein !

Ceci nous conduit à fournir quelques statistiques de tailles de fichiers effectuées à partir de la compilation d'une trentaine de programmes de taille très diverses et faisant pour la plupart appel au GEM :

fichier source .C	:	100 (mesure étalon)
fichier compilé .I	:	100
fichier compilé .1	:	170
fichier compilé .2	:	2
fichier compilé .S	:	150
fichier assemblé .0	:	80
fichier lié .68k	:	310
fichier objet .PRG	:	180

Ces mesures sont naturellement tout à fait dépendantes de la taille des commentaires dans le programme source, du nombre d'appels à des fonctions GEM, de la longueur des programmes, etc. Ces chiffres donnent toutefois des indications sur l'espace mémoire ou sur disque nécessaire pour une compilation ou une édition de liens. Si l'on conserve le fichier source, la place la plus grande est requise à la compilation lors du passage par le générateur de code (C168). Alors doivent cohabiter les fichiers .C, .1, .2 et .S soit un volume souvent supérieur à 4 fois le volume du fichier initial. A l'édition de liens, la coexistence du fichier source .C, du fichier assemblé .O, du fichier lié .68k *et des fichiers temporaires* nécessite un volume souvent supérieur à 6 fois le volume du fichier source. Ainsi l'obtention d'un fichier objet de 70Ko a-t-elle demandée à l'auteur une réservation de 600 Ko de mémoire (en disque virtuel). Au-delà, il faut compiler et lier avec effacement du fichier source, stockage de fichiers de travail sur un autre lecteur de disque... ou disposer d'un disque dur !

3. Une ligne de commande en CP/M-68k peut simplement contenir 126 caractères. Au-delà, soit une recherche de fichier est effectuée sur la base de ces 126 caractères, soit, dans la version la plus récente (LTG), il y a purement et simplement « plantage » du 520. Il ne s'agirait là que d'une anecdote si l'édition de liens ne demandait pas parfois des commandes plus longues ! A ce mal, un seul remède : raccourcir le nom des fichiers liés au minimum possible (une lettre), ce qui permet de lier un peu plus d'une vingtaine de fichiers. Dans l'état actuel, il ne semble pas possible de faire mieux ! Relevons que cette restriction est également valable si l'on utilise un « batch », lequel n'est qu'une simple suite de commandes envoyées précisément vers le «COMMAND.PRG », noyau du CP/M-68k. En fait LINK68 reconnaît les fichiers à appendice 'INP'. On peut donc linker autant de fichiers que nécessaire en les plaçant dans un fichier d'appendice 'INP'.

4.2. Accès aux fonctions GEM en Basic

Au jour où sont rédigées ces lignes le Basic fourni avec le 520 ST ne dispose d'aucune documentation, sinon une liste des instructions, d'ailleurs entachée de fautes d'orthographes et de manques. De plus l'absence quasi totale de traitement d'erreurs rend cet interpréteur pour le moins délicat d'emploi. Essayer d'entrer l'instruction « sound 1000000 » ou, pire, « dir 0: »...

Pourtant ce Basic est bien conçu et offre des possibilités étonnantes, en particulier pour accéder aux fonctions GEM (VDI et AES). Faute de documentation, il a fallu rechercher les informations fournies dans ce chapitre (et au chapitre 5 pour SOUND et WAVE) dans le désassemblage de l'inter-

teur... Ces informations ont ainsi le mérite de pouvoir être garanties authentiques et fonctionnelles ! Il ne faut cependant pas écarter l'éventualité de modifications ultérieures dans l'énoncé ou le mode d'emploi de certaines instructions Basic. On relèvera, par exemple, que la seule et unique variable permettant l'accès aux fonctions AES s'appelle GB (pour GEM Block sans doute), ce nom d'instruction n'apparaissant nulle part dans la liste « officielle » des instructions.

Ces préalables étant posés, il nous a paru nécessaire de fournir quelques indications et exemples sur l'emploi du GEM à partir du Basic, une raison suffisante étant qu'il est ainsi possible au programmeur dans ce langage d'utiliser toutes les informations fournies en Annexe 2. Le programmeur en C ou en assembleur pourra aussi tester telle ou telle fonction à partir du Basic, alternative souvent intéressante compte tenu des temps de compilation et de déblocage.

En Basic, deux instructions permettent d'appeler le GEM :

VDISYS permet d'appeler une fonction VDI.
GEMSYS permet d'appeler une fonction AES.

On notera la confusion de termes pouvant être provoquée par « GEMSYS », qui ne constitue pas un appel générique mais bel et bien de l'AES... Dans les chapitres précédents les méthodes d'appel des fonctions GEM ont été plusieurs fois illustrées (voir aussi l'Annexe 2). Rappelons cependant qu'une fonction GEM nécessite généralement des paramètres d'appel et de retour. Ces paramètres sont stockés dans des tableaux d'entiers (contrl, intin, ptsin, intout, ptsout pour le VDI ; control, global, int_in, int_out, addr_in et addr_out pour l'AES). Pour appeler une fonction GEM à partir du Basic, il sera naturellement nécessaire d'initialiser les paramètres de cette fonction.

Appel de fonction VDI

Pour une fonction VDI, les *variables Basic* suivantes sont utilisées :

CONTRL (10 mots) sert à placer les paramètres fonctionnels.
INTIN (80 mots) pour stock des paramètres d'appel entiers.
PTSIN (128 mots) pour coordonnées en appel (2 entiers).
INTOUT (45 mots) pour stock des paramètres de retour entiers.
PTSOUT (12 mots) pour coordonnées en sortie (2 entiers).

Les variables CONTRL, INTIN, PTSIN, INTOUT, PTSOUT sont des *pointeurs*, autrement dit leur contenu est égal à l'adresse de départ de la zone concernée. Si vous entrez « PRINT CONTRL », il s'affichera dans la fenêtre de sortie un nombre, par exemple 453150. Cela signifie que le tableau

CONTRL commence à cette adresse. Ainsi « PRINT PEEK (453150) » vous fournira la valeur *entière* de CONTRL[0]. Pour placer une valeur x dans CONTRL[5], par exemple, vous entrerez : « POKE x, CONTRL + 10 ». En effet, 5 entiers font 10 octets, l'adresse doit donc être augmentée de 10.

Les zones PTSIN et PTSOUT servent à écrire ou lire des paramètres points, c'est-à-dire des ensembles de deux entiers définissant les coordonnées de ces points. L'annexe 2 fournit pour l'ensemble des fonctions VDI le nombre et le rôle des différents paramètres d'appel. La zone CONTRL est définie comme suit :

CONTRL[0] = numéro de la fonction VDI.
CONTRL[1] = nombre de vecteurs dans PTSIN (un vecteur est un long mot puisqu'il est formé de deux entiers)
CONTRL[2] = nombre de vecteurs dans PTSOUT (cf. ci-dessus).
CONTRL[3] = nombre de paramètres entiers dans INTIN.
CONTRL[4] = nombre de paramètres entiers dans INTOUT.

Ces variables doivent être initialisées avant l'appel d'une fonction VDI.

CONTRL[6] ne doit pas être positionné car il est initialisé une fois pour toutes par l'interpréteur, en principe à la valeur 2. Par contre CONTRL[5] et CONTRL[7] à CONTRL[11] doivent parfois être initialisés (voir chapitre fonction en Annexe 2 pour ces cas particuliers).

Exemples d'appel d'une fonction VDI en Basic :

```
 10 POKE CONTRL,11:'          numéro de fonction VDI
 20 POKE CONTRL+2,2:'         2 vecteurs (PTSIN) en entrée
 30 POKE CONTRL+4,0:'         0 vecteur (PTSOUT) en sortie)
 40 POKE CONTRL+6,0:'         0 entier (INTIN) en entrée
 50 POKE CONTRL+8,0:'         0 entier (INTOUT) en sortie
 60 POKE CONTRL+10,8:'        numéro de la sous-fonction
 70 POKE PTSIN,400:'          abcisse du premier sommet
 80 POKE PTSIN+2,45:'         ordonnée du premier sommet
 90 POKE PTSIN+4,600:'        abcisse du sommet opposé
100 POKE PTSIN+6,150:'        ordonnée du sommet opposé
110 VDISYS:'                  appel de la fonction
```

Ce programme est équivalent à l'appel en C :

vrbox(id_vdi, xysurface)

où xysurface désigne un rectangle défini par ses deux sommets opposés. La sous-fonction 8 provoque le dessin d'un rectangle aux coins arrondis. On relèvera que l'identificateur (CONTRL[6]) n'est pas placé en Basic.

Cet exemple étant un peu particulier (l'appel de sous-fonction est rare), voici un autre exemple plus spectaculaire :

```
10 POKE CONTRL,106:'      numéro de la fonction VDI
20 POKE CONTRL+2,0:'      aucun vecteur en entrée
30 POKE CONTRL+4,0:'      aucun vecteur en sortie
40 POKE CONTRL+6,1:'      un paramètre entier en entrée
50 POKE CONTRL+8,1:'      un paramètre entier en sortie
60 POKE INTIN,9:'         paramètre d'entrée
70 VDISYS:'               appel de la fonction
```

Ce programme correspond à l'appel de la fonction vst_effects, avec pour paramètre d'entrée 9, correspondant à un texte en gras et en italique. Après exécution, vos entrées clavier et les affichages se feront en caractères gras et italiques... jusqu'à un appel de l'éditeur qui rétablit le style de caractère par défaut, c'est-à-dire normal.

Toutes les fonctions du VDI peuvent être appelées de cette façon. Certains appels sont de peu d'intérêt, une instruction correspondante ayant le même effet en Basic (par exemple CIRCLE ou LINEF), d'autres peuvent permettre des effets graphiques tout à fait étonnants. Il est ainsi possible d'effacer tout l'écran par v_clrwk (fonction 3 du VDI), avec passage en mode alphanumérique ! Il faut toutefois savoir que certaines instructions n'ont vraiment d'utilité que dans un programme, l'interpréteur Basic replaçant ses propres paramètres lorsqu'il reprend le contrôle de l'écran.

D'une façon générale, les appels aux fonctions GEM présentent un taux de dangerosité certain, toute erreur dans un paramètre se traduisant souvent par le « plantage » de la machine. Il est donc conseillé de sauver son programme avant de le tester ! Une bonne mesure de prudence consiste aussi à éviter de manipuler CONTRL[6].

Rappelons par ailleurs qu'en Basic ST, le POKE s'effectue normalement sur un *ENTIER*. Il peut être éventuellement pratiqué sur un long mot si l'adresse pokée est suivie de l'appendice « # » (double longueur). Ainsi :

POKE 450146,12	met 0 dans la mémoire d'adresse 450146 et met 12 dans la mémoire d'adresse 450147.
POKE 450146 #,12	met 0 dans les mémoires d'adresse 450146 à 450148 et 12 dans la mémoire 450149.

Appel de fonction AES

L'appel des fonctions AES à partir du Basic est un peu plus délicat que l'appel de fonctions VDI. Aux six zones de paramètres de l'AES ne correspond en effet qu'une seule variable Basic qui, par double accès indirect, autorise toutefois un paramétrage complet. Cette variable GB pointe sur un tableau de pointeurs correspondant aux 6 zones de paramètres. Ainsi :

peek(cdbl (GB)) = adresse de CONTROL.
peek(cdbl (GB+4)) = adresse de GLOBAL.
peek(cdbl (GB+8)) = adresse de INT_IN.
peek(cdbl (GB+12)) = adresse de INT_OUT.
peek(cdbl (GB+16)) = adresse de ADRR_IN.
peek(cdbl (GB+20)) = adresse de ADRR_OUT.

Il est nécessaire d'utiliser « cdbl » pour obtenir le contenu *long* de la mémoire adressée (le suffixe # étant réservé aux nombres). Les zones CONTROL, GLOBAL, INT_IN, INT_OUT, ADRR_IN, ADDR_OUT sont détaillées en Annexe 2. Pour l'appel d'une fonction AES, *contrairement à l'appel d'une fonction VDI,* il est inutile de s'occuper de la zone CONTROL, laquelle est prise en charge par l'interpréteur. Il n'est pas non plus nécessaire de s'occuper de la zone GLOBAL, laquelle n'est pas prise en compte du tout !

L'emploi de certaines fonctions de l'AES est interdit, il s'agit des fonctions appl_init et appl_exit, leur appel risquant d'interférer gravement avec le fonctionnement de l'interpréteur. Toutes les autres fonctions peuvent être utilisées, mais certaines sont très délicates d'emploi et requièrent un soin extrême dans le paramétrage. Il s'agit en particulier des fonctions nécessitant des adresses en paramètres d'appel. Si l'adresse fournie est incorrecte, adieu programme, adieu Basic jusqu'au Reset...

Ces risques étant connus, des perspectives grandioses s'ouvrent au programmeur en Basic, qui peut ouvrir des fenêtres de dialogue, ajouter ou supprimer des objets à celles-ci, afficher des fenêtres d'alerte ou d'erreur, changer les items du menu, utiliser les événements souris ou timer, etc. Voici quelques exemples d'application possibles :

Exemples d'appels d'une fonction AES en Basic

Court mais bon : 10 GEMSYS 53

Ce programme(!) affiche une fenêtre d'erreur avec un numéro aléatoire, en principe 0 mais en fait le contenu de INT_IN|0| lors de l'appel. Pour faire afficher un numéro d'erreur donné, il faut un programme plus long :

```
10 entent = peek(cdbl(GB+8)):'      fournit l'adresse de INT_IN
20 poke entent,12:'                 pourquoi pas l'erreur 12?

30 gemsys 53:'                          affichage de fenêtre d'erreur
```

On notera que la fenêtre est réellement une fenêtre d'erreur, pas un fantôme, ainsi il est nécessaire de cliquer le bouton « OK » pour en sortir ! Cette réalité peut être testée par un programme un peu plus élaboré d'affichage de fenêtre d'alerte (cf. les fonctions AES en Annexe 2, respectivement form_error et form_alert) :

```
10 entent = peek(cdbl(GB+8)):'        fournit l'adresse de INT_IN
20 adrent = peek(cdbl(GB+16)):'       fournit l'adresse de ADDR_IN
30 a$="[1][Voici une fenetre d'alerte][OK|Ah bon!|Annuler]"
40 adchaine = varptr(a$):'            fournit l'adresse de la var. a$
50 poke entent,1:'                    paramètre d'entrée = 1
60 poke cdbl(adrent),adchaine:'       adresse en entrée = adchaîne
70 gemsys 52:'                        affichage de la fenêtre
80 entsort=peek(cdbl(GB+12)):'        paramètre de sortie
90 print"Vous avez pressé le ";peek(entsort);"e bouton"
```

A l'exécution, ce programme provoque l'affichage d'une fenêtre d'alerte avec le texte « Voici une fenêtre d'alerte » et trois boutons : « OK », « Ah bon ! », « Annuler ». Le paramètre d'entrée placé en ligne 50 correspond à la mise par défaut du bouton 1 (« OK »). Les variables entent, adrent et entsort correspondent respectivement à INT_IN, ADRR_IN et INT_OUT. Rappelons enfin que l'instruction Basic VARPTR permet d'obtenir l'adresse d'une variable. Cette instruction est particulièrement utile, conjuguée à l'appel de fonction AES.

Le programme suivant fournit une bonne illustration de l'intérêt d'appeler des fonctions AES en Basic. Il utilise la fonction « evnt_keybd », qui est un peu semblable à un inkey$ et affiche le code clavier et le code ASCII de la touche pressée. La sortie du programme s'effectue par un appui de la touche Return :

```
10 entsort = peek(cdbl(GB+12)):'      adresse de INT_OUT
20 while (boutsort<>13):'             boucle en attente de Return
30 gemsys 20:'                        appel de evnt_keybd
40 boutsort=peek(entsort):'           codes de la touche appuyée
50 codcla=int(boutsort/256):'         masque le code clavier
60 boutsort=boutsort and 255:'        masque le code ASCII
70 print"code:";codcla;"ASCII:";boutsort:'   affichage
80 wend:'                             ferme la boucle while/wend
```

Ces exemples ne sont là que pour vous aider à entreprendre des programmes faisant appel aux fonctions GEM et pour vous mettre l'eau à la bouche ! Une programmation sérieuse de ces fonctions devrait débuter par l'initialisation des variables AES (INT_IN, INT_OUT, ADDR_IN, ADDR_OUT) et peut-être même par l'initialisation des principaux paramètres (ex : ent2ent = peek(cdbl(GB + 8)) qui pointera INTIN[2], etc.

Pour conclure ce sujet, rappelons que les informations ci-dessus pourraient être rendues caduques par une nouvelle version du Basic ou pourraient, plus vraisemblablement, vous sembler bien banales, un manuel sérieux étant disponible entre temps !

4.3. Multitâche et Accessoires

4.3.1. Quel multitâche ?

Le système d'exploitation du 520 ST n'est pas à proprement parler multitâche. On définit en effet comme multitâche un système d'exploitation capable d'assurer le déroulement de plusieurs programmes dans une apparente simultanéité (en fait, les programmes sont exécutés successivement durant de très courts intervalles de temps). Pourtant le GEM dispose de capacités multi-environnement qui valent d'être analysées.

Lors du lancement du système, après un certain nombre d'opérations d'initialisations, est exécuté le programme maître de l'AES. Ce programme débute par une ouverture de poste de travail (v_opnwk), raison pour laquelle, d'ailleurs, il n'est possible ensuite d'ouvrir que des stations virtuelles (v_opnvwk), puis le Bureau (SCRENMGR) et les accessoires éventuels (DESK1, DESK2, DESK3) sont initialisés. Dès lors, l'ensemble de l'activité du 520 est sous le contrôle de l'AES et spécialement d'un programme d'ordonnancement, s'apparentant au « scheduler » bien connu du multitâche.

Le Bureau, les accessoires et l'application éventuellement lancée, disposent tous d'un identificateur (gl_apid de appl_init) et, si un seul programme est actif à un moment donné, l'ordonnanceur, sur la base des identificateurs, effectue en permanence un partage des ressources entre les différentes applications. C'est ce partage qui vous autorise, par exemple, à ouvrir un accessoire dans le cours d'un programme.

La plupart des événements sont toutefois gérés sur le mode des interruptions. Ainsi, lorsque vous déplacez la souris, ce déplacement, s'il est suffisamment significatif (cf. 5.2. commande $B), provoque une interruption clavier. Le traitement de celle-ci se résume à un enregistrement de la nouvelle position de la souris. A l'interruption de balayage vidéo suivante, l'activité de la souris est testée et, si elle est active et s'il y a eu changement de position depuis la dernière interruption de balayage, le pointeur de souris est modifié à l'écran.

Dans ce cas simple et particulier, seule l'activité ou non de la souris dans l'application active est testée. La procédure est plus complexe si l'événement à traiter par l'ordonnanceur est un « message », par exemple un cliquage dans la case de fermeture d'une fenêtre. Dans un tel cas, l'activité de la fenêtre est testée. Si la fenêtre est active, il y a fermeture avec éventuellement fin de l'application si celle-ci ne possédait qu'une fenêtre. Sinon, le cliquage dans la case de fermeture est assimilé à une demande de réactivation, d'où passage d'une application à une autre si la fenêtre cliquée n'appartient pas à l'application active (cas d'un accessoire, par exemple).

Les interactions entre le Bureau et l'utilisateur (le Bureau étant constitué de l'ensemble des éléments extérieurs à la fenêtre active, autrement dit la barre des menus, les cases de fermeture, de réajustement de taille, etc.) sont contrôlées par le programme principal de l'AES (SCRENMGR, gestionnaire d'écran).

Les instructions GEM appl_read, appl_write, appl_find et l'ensemble des fonctions « Shell » (cf. Annexe 2) autorisent une certaine forme de communication entre diverses applications, des informations pouvant être envoyées de l'une à l'autre. Il s'agit d'une implémentation très primitive du canal « PIPE » des systèmes multitâches, ouvrant tout de même quelques perspectives intéressantes, l'envoi de messages (codes de touches clavier, opérations de manipulations de fenêtres) étant ainsi possible.

Selon une expression désormais courante, le « couper/coller » est également possible entre plusieurs applications. Il s'opère par le biais du « clipboard » (que nous ne traduirons pas par « presse-papiers » !) et de fichiers « SCRAP ». Les instructions AES permettant d'utiliser ce « clipboard » sont présentées au chapitre 2 (scrp_read et scrp_write). On notera que le passage d'informations d'une application à une autre par ce moyen oblige à des accès disquettes, ce qui garantit du moins la pérennité de ces informations mais s'avère un procédé assez lourd, en particulier lorsque l'on désire « couper/coller » avec un accessoire...

4.3.2. Accessoires

Les accessoires constituent des applications comme les autres, comme le gestionnaire d'écran ou le traitement de texte. Leur particularité tient à ce que leur contrôle peut être effectué à partir de la barre des menus (option « Bureau »).

Seuls trois *programmes* accessoires peuvent fonctionner concurremment à une application, cette limitation étant due à des impératifs de limitation des zones de mémoire système (chaque application possède son en-tête, ses tampons, ses drapeaux). Ces trois programmes peuvent se partager jusqu'à six accessoires, le partage pouvant être effectué de façon quelconque (il est même envisageable de donner le contrôle des six accessoires à un seul programme). La limitation à six accessoires est due à un choix préétabli de la barre des menus, dont l'option « Bureau », contrôlée par le gestionnaire d'écran, ne peut comporter que six items dédiés aux accessoires.

On peut regretter que les concepteurs du GEM n'aient pas prévu la présence simultanée de plus de 6 accessoires... Rappelons toutefois qu'il est toujours possible, par des manipulations assez complexes des objets du Menu,

de modifier le titre d'un accessoire, ce qui permet, si le *programme* accessoire est prévu pour cela, de gérer plusieurs sous-programmes pour un même numéro d'accessoire.

Comment se fait-il qu'un même programme puisse contrôler plusieurs accessoires indépendants, donc plusieurs applications ? A cette question, le désassemblage du système, spécifiquement de l'AES, permet de répondre ainsi :

— Un accessoire n'est pas une application. Seul le *programme* accessoire constitue une application, c'est-à-dire une entité pourvue d'un identificateur d'application et de caractéristiques graphiques indépendantes.
— L'initialisation de chacun des items accessoires du Menu est effectuée par le programme accessoire (cf. menu_register).
— Le programme accessoire, une fois l'initialisation opérée, reste *toujours actif*. Il boucle dans une attente de message (evnt_mesag).

Cette boucle d'attente est toutefois une fausse boucle, comme toutes les fonctions GEM d'attente d'événement. En fait, lorsque l'on utilise une fonction « evnt » de l'AES, on redonne la main à l'ordonnanceur en lui signalant que l'on attend tel ou tel type d'événement. L'ordonnanceur examine alors si d'autres applications ou accessoires sont suspendus et s'ils doivent donc être réactivés et, dans le cas contraire, attend l'événement suivant et selon la nature de ce dernier réactive telle ou telle application en attente.

Pour en revenir au cas particulier de l'accessoire, l'attente au départ est celle d'un message. Si un message arrive, son numéro est comparé à « AC.OPEN » et, si c'est le message reçu, le quatrième mot du message est comparé à l'item du Menu correspondant à chaque accessoire du programme. S'il y a correspondance des numéros, alors l'accessoire concerné est activé. Cet accessoire peut alors ouvrir une station de travail virtuelle, une fenêtre, etc., comme toute application. Naturellement, lors de la sortie de l'accessoire, la station de travail et la (ou les) fenêtre(s) doivent être refermées.

Si l'on désire créer un ou plusieurs accessoires, la différence entre accessoire et programme accessoire doit être parfaitement comprise, faute de quoi l'on s'expose à de nombreux déboires. Ainsi le passage d'informations à partir d'une application ne peut se pratiquer que vers un *programme* accessoire, quitte à ce que ce programme redirige ensuite ces informations vers tel ou tel accessoire. Il faut reconnaître que la meilleure façon de créer des accessoires consiste à faire coïncider un accessoire et un programme accessoire...

En matière d'accessoire, tout ou à peu près est réalisable, une fois connues les règles de maniement décrites ci-dessus. Ainsi, bien que le démarrage normal d'un accessoire s'effectue lorsque survient un message d'acti-

vation (AC OPEN), rien ne s'oppose à ce que soit un evnt_timer ou un evnt_multi qui contrôle l'accès à tel ou tel accessoire. A titre indicatif, l'ouverture de l'accessoire « panneau de contrôle » s'effectue par un message « AC OPEN » avec identification du numéro d'item correspondant à « panneau de contrôle » dans la barre des menus, mais la « boucle » du programme accessoire est en fait un evnt_multi, à la fois message et timer. C'est l'événement timer qui provoque la remise à l'heure de l'horloge toutes les minutes, permettant ainsi de disposer de l'heure sur le Bureau, à partir du moment où le panneau de contrôle a été ouvert et même si cet accessoire n'est plus activé.

Par un procédé identique à celui utilisé pour le panneau de contrôle, il serait possible, par exemple, de déclancher l'ouverture d'un accessoire ou d'une simple fenêtre à une heure donnée (fonction alarme). Sans entrer dans les détails, la question étant abordée au chapitre 2 et en annexe 2, signalons pour conclure que les interactions entre les affichages et, d'une façon générale, les opérations graphiques, d'une application et des divers accessoires sont à l'origine de nombreux problèmes internes de fonctionnement des programmes. Tout test de programme devrait, en conséquence, comprendre plusieurs tests relatifs aux accessoires : ouverture simultanée à l'application de plusieurs accessoires, déplacement de la fenêtre du panneau de contrôle ou de l'application, avec ouverture et fermeture et surtout avec simple désactivation de chacun des accessoires.

5. PROGRAMMATION DES ENTRÉES-SORTIES

L'immense majorité des fonctions GEM et des fonctions système, lesquelles sont détaillées en annexes, sont consacrées aux entrées-sorties, affichage graphique pour le GEM et opérations élémentaires ou traitement de fichiers pour les fonctions système (traps). La compréhension et l'utilisation de ces fonctions devrait normalement suffire à tout programmeur pour résoudre un quelconque problème d'entrée-sortie.

Dans la pratique certaines fonctions ressemblent à la mère Gigogne... Ainsi de la commande de sortie du synthétiseur de son et surtout de la fonction d'envoi de commandes au microprocesseur clavier, cette dernière autorisant par exemple la lecture ou l'écriture de mémoires du 6301 et même la demande d'exécution de routines internes à ce processeur... Enfin la question des interruptions nous paraît mériter quelques développements.

5.1. Les interruptions

Le 68000 supporte 7 niveaux d'interruption, par le biais de trois lignes. Ces interruptions sont masquables en plaçant les bits 8 à 10 du registre d'état à un. Sur le 520 ST, seuls deux niveaux d'interruption sont vectorisés : les niveaux 2 et 4, correspondant aux interruptions horizontale et verticale de balayage vidéo.

L'interruption de niveau 2 (fréquence de ligne) est masquée par le système à l'initialisation. Étant donné qu'un programmeur ignorant pourrait la démasquer, par exemple en replaçant à 0 les bits 8 à 10 du registre d'état, une routine de traitement existe. Cette routine ne fait qu'une chose : elle replace le

masque dans le registre d'état et inhibe donc de nouveau l'interruption de niveau 2 !

L'interruption de niveau 4 (fréquence de trame) est par contre fort utilisée par le TOS. Elle n'est masquée que fort rarement, lors de modifications liées à l'écran (envoi de commandes au contrôleur vidéo) ou lors de « portions critiques » du code (changement de mode du 68000, initialisation d'un vecteur d'interruption 68901, etc.). Cette interruption est provoquée par le contrôleur vidéo à chaque balayage vertical, c'est-à-dire 70 fois par seconde avec un moniteur monochrome et 60 fois par seconde avec un moniteur couleur.

La routine vectorisée d'interruption de niveau 4 exécute l'ensemble des tâches suivantes :

— incrémentation du compteur d'interruptions verticales ($466.L).
— test du drapeau de prise en compte des interruptions verticales ($452) et incrémentation du compteur $466.L *si le drapeau est à 1*. Ce drapeau doit être placé à 0 lorsque l'on change la base de la mémoire d'écran.
— test de moniteur (à travers le bit 7 de $FFFA01) et modification éventuelle de la résolution vidéo avec départ chaud.
— test de l'état du curseur (actif ou non) et affichage ou effacement selon le nombre de périodes écoulées depuis le dernier changement.
— test du pointeur de palette ($45A) et implantation éventuelle d'une nouvelle palette de couleurs.
— test du pointeur de base écran ($45E) et changement de base si ce pointeur est non nul.
— desélection des lecteurs de disques si non verrouillés (toutes les huit interruptions verticales).
— exécution de chacune des routines d'une liste dont l'adresse de départ est contenu dans $456.
— test du drapeau de copie d'écran ($4EE) et « spooling » d'imprimante si le drapeau est à 0.

La partie la plus intéressante et la plus accessible au programmeur de cette routine d'interruption est sans conteste l'exécution des routines pointées par $456. A cette adresse est placé un pointeur vers une liste de pointeurs. Cette liste de pointeurs débute normalement en $4CE (valeur contenue dans $456). A l'ouverture d'une station de travail (v_opnwk), une adresse de routine est placée dans cette liste (traitement du retraçage de la souris).

Pour faire exécuter une routine à chaque interruption verticale, il suffit de demander le contenu de $456 puis, à partir de l'adresse de base contenue dans $456, de tester chacun des longs mots et de placer l'adresse de sa routine à la place du premier long mot nul. La liste peut contenir au maximum

adresses de routines (elle est en fait de 8 éléments mais le dernier doit être [illegible]).

Interruptions 68901

Le circuit multi-fonctions de gestion des périphériques 68901 gère la plupart des périphériques et par ce fait les interruptions que ceux-ci provoquent. Ces interruptions sont répercutées vers le 68000 par le biais de seize vecteurs ([illegible] à $F) provoquant l'exécution des routines vectoriées $100 à $13C. Chacun de ces vecteurs à la priorité (pour le 68901) et la fonction suivante :

vecteur	priorité	fonction
$100	basse	ligne Busy de l'interface parallèle.
$104		dcd de l'entrée-sortie série (ligne 8).
$108		cts de l'entrée-sortie série (ligne 5).
$10C		contrôleur vidéo (fin d'opération)
$110		timer D (horloge de cadencement RS 232).
$114		timer C (horloge système à 200 Hz).
$118		ACIA 6850 (MIDI et clavier).
$11C		contrôleur de disque et de DMA.
$120		compteur d'interruptions horizontales.
$124		erreur à l'émission RS 232.
$128		tampon d'émission RS 232 plein.
$12C		erreur de réception RS 232.
$130		tampon de réception RS 232 plein.
$134		timer A (disponible au programmeur).
$138		indicateur de sonnerie RS 232 (ligne 22).
$13C	haute	détection de moniteur monochrome.

Parmi ces interruptions, seules sont implementés dans le système de base les vecteurs suivants :

$114 Interruption horloge système 200 Hz.

Sont effectuées les opérations suivantes :

— incrémentation du compteur $4BA.L
— si ce compteur est divisible par 4, donc à la fréquence de 50 Hz, la suite de la routine est exécutée. Sinon, retour d'interruption.
— recherche de commande son éventuelle et envoi (cf. 5.3.).
— test de l'auto-répétition des touches et si auto-répétition active, décrémentation du compteur et stockage éventuel de la touche appuyée si le délai d'auto-répétition est épuisé.
— exécution de la routine timer du gestionnaire d'événements AES (cf. Annexe 2 : evnt_timer).

— désactivation du drapeau d'interruption 68901.

$118 Interruption ACIA MIDI et clavier.
Chacun des registres de contrôle ACIA dédié à l'interface MIDI et à l'interface clavier est testé et, selon le circuit initiateur de l'interruption, soit l'entrée MIDI soit l'entrée clavier est prise en compte. La routine de traitement d'entrée sur l'interface MIDI se résume à un stockage de l'octet reçu dans le tampon MIDI (la fonction TRAP 14, $E, permet de connaître l'adresse de départ de ce tampon), avec mise à jour des pointeurs de tampon.

La routine d'entrée clavier fait l'objet d'une analyse détaillée en 5.2. Notons cependant ici que l'ensemble des sous-routines traitant les interruptions MIDI et clavier peuvent être modifiées par le programmeur. Il suffit d'appeler la fonction $22 du Trap 14 pour obtenir l'adresse de base de la liste des pointeurs suivants :

0.L	entrée MIDI	voir ci-dessus pour information.
4.L	erreur clavier	non prise en compte (RTS).
8.L	erreur MIDI	non prise en compte (RTS).
C.L	état clavier	adresse de la routine de stockage et de traitement de l'état du clavier (cf. 5.2., commande $F6)
10.L	position souris	adresse de la routine de stockage et de traitement de la position et des boutons de la souris.
14.L	horloge	stockage de la date et de l'heure.
18.L	manettes de jeu	adresse de la routine de stockage et de traitement des événements manettes de jeu.

Les quatre routines ci-dessus seront analysées en 5.2.

$124 Erreur d'émission RS 232.

L'octet du registre de contrôle de l'émetteur série est stocké dans $1D du tampon RS 232. L'interruption est désactivée.

$128 Tampon d'émission RS 232 plein.

En fonction du protocole (CTS/RTS ou XON/XOFF), un signal est envoyé ou non demandant l'autorisation de transfert. L'interruption est désactivée.

$12C Erreur de réception RS 232.

L'octet du registre de contrôle du récepteur série est stocké dans $1C du tampon RS 232. L'interruption est désactivée.

$130 Tampon de réception RS 232 plein.

En fonction du protocole, un signal de RTS ou un XON/XOFF est envoyé pour arrêt de la transmission. L'interruption est désactivée.

Le vecteur $108, correspondant à CTS, n'est initialisé par le système que si le protocole lui correspondant est validé. D'une façon générale, tous les vecteurs peuvent être modifiés, d'autant que la fonction Mfpint (Trap 14, SD) est dédiée à ce seul usage. L'interruption timer A est cependant la plus intéressante à implanter (cf. chapitre 1 et fonction Xbtimer (Trap 14, $1F). Il convient, pour tout changement des vecteurs d'interruption, de désactiver ce vecteur avant modification puis de le réactiver, ceci au moyen des fonctions Jdisint (Trap 14, $1B) et Jenabint (Trap 14, $1C).

5.2. Clavier, souris, horloge et manettes de jeu

Tous ces périphériques sont gérés par le microprocesseur 6301, lequel génère des interruptions vers le 68901, qui les répercute au 68000. Mais il est également possible d'envoyer des demandes d'information ou des commandes vers le 6301. Les périphériques clavier, souris, horloge et manettes de jeu représentent, en dehors des disques, les périphériques les mieux pourvus en fonctions système. En voici la liste :

Cconin	Trap 1, $1	attente d'entrée clavier avec affichage.
Crawio	Trap 1, $6	attente d'entrée « brute » sans affichage.
Cnecin	Trap 1, $8	attente d'entrée clavier sans affichage.
Cconrs	Trap 1, $A	attente de chaîne entrée au clavier.
Cconis	Trap 1, $B	demande d'état du tampon d'entrée clavier.
Bconstat	Trap 13, $1	(param = 2) demande état du clavier en entrée.
Bconin	Trap 13, $2	(param = 2) attente d'entrée clavier.
Bconout	Trap 13, $3	(param = 4) envoi d'un octet vers le clavier.
Bcostat	Trap 13, $8	(param = 4) demande d'état du clavier en sortie.
Getshift	Trap 13, $B	activation d'une touche spéciale du clavier.
Initmous	Trap 14, $0	initialisation de la souris.
Keytbl	Trap 14, $10	fixe les adresses des tables de décodage touches.
Bioskeys	Trap 14, $18	replace les adresses système tables de décodage.
Ikbdws	Trap 14, $19	envoi de chaîne vers le clavier (commande).
Kbdvbase	Trap 14, $22	demande le pointeur des vecteurs clavier.
Kbdrate	Trap 14, $23	fixe la fréquence d'auto-répétition des touches.

Certaines de ces fonctions font double emploi, par exemple Cnecin et Bconin. Il n'a pas été fait état ici des fonctions liées au temps (lecture ou écriture de la date et de l'heure), lesquelles dépendent toutes de l'horloge du 6301 et des interruptions horloge qu'il génère. Les fonctions d'entrée clavier étant simples et semblables aux fonctions équivalentes sur les micro-ordinateurs, ce sous-chapitre sera consacré aux sorties clavier, c'est-à-dire à l'envoi de commandes vers le 6301.

L'envoi d'une commande vers le 6301 s'effectue très simplement, en C comme en assembleur, à l'aide de la fonction Ikbdws (Trap 14, $19). Il est possible d'envoyer plusieurs commandes au 6301 dans une même chaîne mais cela ne paraît pas très recommandé pour la lisibilité ultérieure du programme.

Pour une utilisation ponctuelle, en C comme en assembleur, on pourra réserver un tableau de caractères (que nous nommerons ici com_cla) pour le stockage des commandes (7 octets par commande au maximum) et un entier pour le stockage du nombre d'octets à envoyer (que nous nommerons ici lcom_cla). L'appel se fera ensuite par :

```
int   lcom_cla;
char  com_cla[7];

Ikbdws( lcom_cla, com_cla );     en langage C
```

et en assembleur par :

```
move.l  com_cla,-(a7)        * stocke le contenu de lcom_cla   *
move.w  #lcom_cla,-(a7)      * stocke l'adresse de com_cla     *
move.w  #$19,-(a7)           * stocke le numero de la fonction *
trap    #14                  * exécute Ikbdws                  *
addq.l  #8,a7                * restaure le pointeur de pile    *
```

Les pages qui suivent fournissent la fonction et les conditions d'emploi de chacune des commandes pouvant être envoyées au circuit clavier 6301. Les commandes provoquent des actions mais également un stockage des paramètres dans la RAM du 6301. Les adresses RAM affectées et la valeur par défaut sont fournies en fin de commande.

Commande $80, Reset

Commande à securité, doit être suivie de l'envoi de $1 pour être validée. L'horloge n'est pas modifiée (elle est initialisée une fois pour toutes lors du Reset du 6301, parallèle au Reset du système). Le 6301 renvoie, lors d'une commande reset, le numéro de version de la ROM, actuellement $F0. Toute touche enfoncée lors du reset provoque l'envoi par le clavier du code de la touche avec le bit 7 à 1.

Commande $07. Mode de traitement des boutons de la souris

Doit être suivie d'un octet. Les bits 3 à 7 de cet octet doivent être nuls. Si le bit 2 est à 1, les boutons de la souris sont traités comme des touches du clavier. Sinon, si le bit 1 est à 1, le relâchement du bouton provoque l'envoi de la position absolue de la souris et, si le bit 0 est à 1, la pression sur le bouton entraîne l'envoi de la position absolue de la souris.
(mode stocké dans $B4, valeur par défaut : 0).

Commande $08. Déplacements de souris relevés en mode relatif

Le clavier envoie la position de la souris lorsque le déplacement de celle-ci par rapport à sa position antérieure dépasse une base fixée par la commande $B (déplacement horizontal ou vertical)
(mode par défaut, $BC et $BD annulés, $C9 positionné).

Commande $09. Déplacements de souris autorisés en mode absolu

Cette commande doit être suivie de 4 octets. Les deux premiers définissent la position horizontale maximale et les deux suivants la position verticale maximale.
(posit. hor. : $AA et $AB, posit. vert. : $AC et $AD, nuls par défaut).

Commande $0A. Place la souris en mode clavier (!)

Suivie de deux octets définissant la distance horizontale puis verticale minimales pour que soit envoyée le code de [LEFT], [RIGHT], [UP] ou [DOWN]. Remplace le positionnement relatif ou absolu. Le clavier envoie les codes équivalents à un appui des touches fléchées lors de déplacements de la souris excédant les intervalles fixes par rapport à la position antérieure.
(int. hor. : $AE, int. vert. : $AF, $BA et $BB annulés).

Commande $0B. Place les intervalles de déplacement de la souris

Suivie de deux octets indiquant les intervalles horizontaux et verticaux minimum pour un envoi de la position de la souris au 68000 (uniquement valable en mode de déplacement relatif).
(int. hor. : $B0, int. vert. : $B1, $BC et $BD annulés, intervalles à 1 par défaut).

Commande $0C. Fixe l'échelle de déplacement de la souris en mode absolu

Suivie de deux octets indiquant les facteurs d'échelle horizontale et verticale lors de positionnement absolu.
(échelle hor. : $B2, échelle vert. : $B3, 1 par défaut).

Commande $0D. Demande de la position de la souris

Uniquement valable si la souris est en mode de positionnement absolu. Le 6301 renvoie la position suivant le code suivant :

SF7	en-tête de la souris en positionnement absolue.
octet	bit 0 : à 1 si bouton droit pressé depuis précédente demande
	bit 1 : à 1 si bouton droit relâché depuis précédente demande
	bit 2 : à 1 si bouton gauche pressé depuis précédente demande
	bit 3 : à 1 si bouton gauche relâché depuis précédente demande

2 octets indiquant la position horizontale
2 octets indiquant la position verticale
(état des boutons dans $C2).

Commande $0E. Fixe la position absolue de la souris

Suivie d'un octet nul puis de la position horizontale et de la position verticale, chacune sur 2 octets. Place la souris en mode absolu.
(pos. hor. : $B6-$B7, pos. vert. : $B8-$B9).

Commande $0F. Place l'origine verticale en bas.

Valable en mode absolu et relatif. Les mouvements de la souris vers l'utilisateur deviennent négatifs et positifs dans l'autre sens.
(bit 0 de $C9 à 1).

Commande $10. Place l'origine verticale en haut

Valable dans les deux modes. Mouvements inverses par rapport à la commande précédente.
(bit 0 de $C9 à 0. Position par défaut).

Commande $11. Réactivation des transmissions

Inverse de la commande de Pause ($13).

Commande $12. Désactive la souris

Ne désactive que le mode relatif ou absolu, pas le mode clavier (cf. $A). Réactivation possible par les commandes $8, $9 et $A.

Commande $13. Désactivation des transmissions (Pause)

Transmissions stoppées jusqu'à ce qu'une commande valide soit reçue. Les événements continuent à être stockés dans le tampon de 8 octets du 6301.

Commande $14. Mode événement des manettes de jeu

Provoque l'envoi d'une commande par le clavier à chaque événement (ouverture ou fermeture de contact). Active la manette. La commande envoyée par le clavier tient en 2 octets : le premier égal à $FE si manette 0, à $FF si manette 1, le second fournit la position sur les bits 0 à 3 et l'appui du bouton de « feu » par le bit 7.
(mode par défaut).

Commande $15. Mode interrogation des manettes de jeu

L'envoi de l'état des manettes de jeu n'est fourni que sur demande par une commande $16. Active la manette.

Commande $16. Interrogation des manettes de jeu

Demande l'état des manettes de jeu. Le 6301 répond par l'octet $FD suivi de 2 octets définissant pour la manette 0 et la manette 1 l'appui du bouton de « feu » (bit 7 à 1) et la position (bits 0 à 3).

Commande $17. Place le 6301 en mode manettes de jeu

Dans ce mode, le 6301 contrôle uniquement l'horloge et les manettes de jeu. Cette commande doit être suivie d'un octet indiquant la période d'interrogation des manettes en centièmes de seconde. A chaque période ainsi définie, le 6301 envoie 2 octets, le premier définissant l'état des boutons de feu des manettes (bit 1 pour manette 0, bit 0 pour manette 1, autres bits nuls). Dans le second octet, les bits 4 à 7 définissent la position de la manette 0 et les bits 0 à 3 la position de la manette 1.

Commande $18. Place le 6301 en mode boutons de « feu ».

Dans ce mode, le 6301 contrôle uniquement l'horloge et le bouton de « feu » de la manette de jeu 1. Retourne continûment l'état du bouton dans chacun des bits d'une série d'octets. 781 octets, correspondant à 6 250 positions, sont envoyés chaque seconde.

Commande $19. Place la manette de jeu 0 en mode clavier

Suivie de six octets définis comme suit :

RX : période, en dixièmes de seconde, avant que le point de déplacement horizontal soit atteint.

RY : période, en dixième de seconde, avant que le point de déplacement vertical soit atteint.

TX : période, en dixièmes de seconde, de fermeture du contact avant qu'un déplacement horizontal soit signalé.

TY : période, en dixièmes de seconde, de fermeture du contact avant qu'un déplacement vertical soit signalé.

VX : période, en dixièmes de seconde, de fermeture du contact avant qu'un déplacement horizontal soit signalé, après écoulement de RX.

VY : période, en dixièmes de seconde, de fermeture du contact avant qu'un déplacement vertical soit signalé, après écoulement de RY.

Dans ce mode, la manette de jeu 0 a un effet équivalent aux touches du curseur.
($94, $A1 à $A3 annulés).

Commande $1A. Désactive les manettes de jeu

Les manettes de jeu peuvent être réactivées par les commandes $14, $15, $17, $18 et $19.

Commande $1B. Fixe la date et l'heure

La date et l'heure doivent suivre la commande sur 6 octets (année sur 2 chiffres significatifs, mois, jour, heure, minutes, secondes). Toutes ces données doivent être en BCD (binaire code décimal). Les chiffres non envoyés en BCD ne sont pas pris en compte.
(Place date et heure dans $82 à $87).

Commande $1C. Demande la date et l'heure

La date et l'heure courantes sont retournées sur 6 octets (définis comme ci-dessus) précédés de l'octet $FC (commande horloge).

Commande $20. Écriture de données en mémoire vive du 6301

Suivie de l'adresse de départ (sur 2 octets) où stocker les données, puis d'un octet indiquant le nombre de données à écrire, puis des données.

Commande $21. Lecture de données en mémoire du 6301.

Suivie de l'adresse de départ (sur 2 octets) à partir de laquelle lire les données. Le 6301 répond par les octets $F6 et $20 de prise en compte de la commande puis par 6 octets de données débutant à l'adresse fournie.

Commande $22. Demande d'exécution d'une routine du 6301

Suivie de l'adresse de la routine à exécuter (sur 2 octets).

Commandes $87 à $8C, $8F, $90, $92, $94, $95, $99 et $9A

Ces commandes sont des demandes d'état correspondant aux routines $07 à $0C, $0F, $10, $12, $14, $15, $19 et $1A. Le 6301 répond par un octet de statut ($F6 à $FF) puis sept octets fournissant l'état (avec des octets nuls comme remplissage si nécessaire). Ces octets de statut correspondent précisément aux octets envoyés lors des commandes précédentes :

commande $87 mode action des boutons de la souris (cf. commande 7).
commandes $88 à $8A : un octet de mode (8 = relatif, 9 = absolu A = clavier), puis 6 octets nuls si mode relatif, 2 octets d'intervalles horiz. et vertical si mode clavier puis 4 octets nuls, 4 octets de position horizontale et verticale et 2 nuls si mode absolu.
commande $8B intervalles horizontal et vertical et 5 octets nuls.
commande $8C échelles horizontale et verticale et 5 octets nuls.
commandes $8F et $90 : retourne $F si Y en bas, $10 si Y en haut.
commande $92 souris active ($0), désactivée ($12).
commande $94 à $95 et $99 : mode manettes de jeu.
commande $9A manettes actives ($0), désactives ($1A).

Les demandes d'état ne sont pas prises en compte si le 6301 se trouve en mode manettes de jeu ou en mode bouton de « feu » (commandes $17 et $18).

La commande $08 de déplacement relatif de la souris (et les commandes $09 à $0C) font intervenir la notion de déplacement de la souris. Ce déplacement est repéré sur une échelle interne au 6301 et comptant environ 80 unités par centimètres parcouru. La scrutation de la souris est effectuée à une fréquence autorisant des vitesses de déplacement de l'ordre de 25 cm/s (0,9 km/h pour les fanas de la vitesse !). En mode relatif, la position de la souris est envoyée au 68000 chaque fois qu'elle dépasse un intervalle horizontal ou vertical par rapport à sa position antérieure ou lorsque l'état des boutons est modifié. En mode absolu, il faut interroger le 6301 pour connaître la position de la souris. En mode clavier, selon le facteur d'échelle, le 6301 envoie les codes de touches correspondant aux déplacements (les appuis des boutons de la souris provoquent dans ce cas l'envoi des codes $74 et $75 au 68000, codes clavier, cf. Cconin, Trap 1, fonction $1).

Le microprocesseur 6301 dispose de vingt octets de mémoire vive pour le contrôle des entrées-sorties et des interruptions. Ces mémoires, d'adresse 0 à $13, ont la fonction suivante :

00 registre de direction du port 1.
01 registre de direction du port 2.
02 registre d'adresse du port 1.
03 registre d'adresse du port 2.
04 registre de direction des données.
05 registre de direction des adresses.
06 adresses des 8 lignes de données.
07 adresses des 8 lignes d'adresses A8 à A15.
08 registre de contrôle et d'état du timer 6301.
09-0A compteur de cycles machine (incrémenté toutes les μs).
0B-0C registre de comparaison avec le précédent.
0D-0E registre de saisie des entrées timer.
0F registre de contrôle du port 3.
10 registre de mode et de vitesse de l'interface série rapide.
11 registre de contrôle et d'état de l'interface série rapide.
12 registre de lecture des données en série rapide.
13 registre d'écriture des données en série rapide.

Ces registres constituent la zone de contrôle du 6301, lequel dispose par ailleurs de 128 octets de RAM (d'adresses $80 à $FF) et de 4 Ko de ROM (d'adresses $F000 à $FFFF).

La carte mémoire vive du 6301 est la suivante :

$80-$81	compteur de millisecondes (décrémenté de $3E8 à 0).
$82	année (en BCD, sur 2 chiffres inférieurs).
$83	mois, en BCD.
$84	jour, en BCD.
$85	heures, en BCD.
$86	minutes, en BCD.
$87	secondes, en BCD
$88	drapeau d'initialisation (à $AA si Reset effectué).
$89-$94	mémoires de travail pour interruptions.
$95	RX (voir commande $19).
$96	RY (voir commande $19).
$97	TX (voir commande $19).
$98	TY (voir commande $19).
$99	VX (voir commande $19).
$9A	VY (voir commande $19).
$9B-$9C	mémoires de travail pour scrutation du clavier.
$9D	0 si mode bouton de « feu », sinon compteur interruptions.
$9E-$9F	mémoires de travail si mode manettes de jeu.
$A0-$A3	mémoires de travail si manettes en mode clavier.
$A4	état courant de la manette 0.
$A5	état courant de la manette 1.
$A6	période de temporisation mode bouton de « feu ».
$A7	période de temporisation mode manettes de jeu.
$A8-$A9	mémoires de travail si manettes en mode clavier.
$AA-$AB	position horizontale maximum de la souris (comm. $9).
$AC-$AD	position verticale maximum de la souris (comm. $9).
$AE	distance horizontale souris en mode clavier (comm. $A).
$AF	distance verticale souris en mode clavier (comm. $A).
$B0	intervalle horizontal souris en mode relatif (comm. $B).
$B1	intervalle vertical souris en mode relatif (comm. $B).
$B2	échelle horizontale de déplacement de souris (comm. $C).
$B3	échelle verticale de déplacement de souris (comm. $C).
$B4	mode de traitement des boutons de la souris (comm. $7).
$B5	mémoire de travail (remplissage commande $E).
$B6-$B7	coordonnée horizontale de la souris (comm. $E).
$B8-$B9	coordonnée verticale de la souris (comm. $E).
$BA-$BB	mémoires de travail si souris en mode clavier.
$BC-$BD	mémoires de travail si souris en mode relatif.
$BE-$C8	mémoires de travail pour calcul des données à envoyer.
$C9	bit 0 : à 1 si Y en bas, à 0 si Y en haut (comm. $10, $11). bit 1 : à 1 si commande manette, à 0 si commande souris.

	bit 2 : inutilisé.
	bit 3 : à 1 si pause avec empilement des données.
	bit 4 : à 1 si souris en mode relatif.
	bit 5 : à 1 si souris en mode absolu.
	bit 6 : à 1 si souris en mode clavier.
	bit 7 : à 1 si souris active.
$CA	bit 0 : à 1 si en mode bouton de « feu ».
	bit 1 : à 1 si 6301 actif (commande $11).
	bit 2 : à 1 si manettes en mode clavier.
	bit 3 : à 1 si manettes en mode événement.
	bit 4 : à 1 si manettes en mode interrogation.
	bit 5 : à 1 si manettes actives.
	bits 6 et 7 : inutilisés.
$CB	bits 0 à 2 : compteur d'octets reçus (0 à 7).
	bit 3 : à 1 si Pause (comm. $13).
	bit 4 : drapeau d'erreur « Over-run framing ».
	bit 5 : à 1 si tampon de réception plein.
	bit 6 : à 1 si tampon de réception non vide.
	bit 7 : à 1 si octet en cours de réception.
$CC	compteur d'octets du tampon de réception.
$CD-$D3	TAMPON DE RÉCEPTION DES DONNÉES 68000 (7 octets).
$D4	compteur d'interruptions non prises en compte.
$D5	compteur de boucle pour stockage dans le tampon.
$D6	compteur de numéro d'octet à envoyer.
$D7	nombre d'octets à émettre.
$D8	compteur de comparaison avec nombre octets à émettre.
$D9-$ED	TAMPON D'ÉMISSION VERS LE 68000 (20 octets).
$EE-$EF	stockage d'adresse (commandes $20 et $21).
$F0	stockage nombre octets à recevoir (commande $20).
$F1-$FF	PILE du 6301.

Ces informations pourraient paraître superflues si la commande $21 ne permettait de lire des données dans la mémoire vive du 6301. Il est ainsi possible, par exemple, de disposer d'un compteur au millième de seconde (compteur situé en $80-$81, réinitialisé à $3E8 chaque seconde et décrémenté chaque milliseconde). Il suffit pour cela d'appeler :

Ikbdws (2, com_cla) avec com_cla[0] = $21,

La réponse est alors obtenue par lecture des sorties clavier, les deux premiers octets étant des codes de commandes ($F6 et $20), les deux octets suivants fournissant la valeur du compteur et les quatre suivants pouvant être négligés (contenu des mémoires $82 à $85).

Il est même possible de disposer d'un compteur de millionièmes de seconde (situé en $9 et $A, remis à $3E8 chaque millième de seconde et décrémenté), mais ce compteur est pratiquement inutilisable, la vitesse de transmission série rapide (7 812,5 bits/s) étant trop faible...

La commande $20 autorisant l'écriture en mémoire vive du 6301, il est possible de reconfigurer tous les paramètres de gestion de la souris et des manettes en modifiant le contenu des mémoires $95 à $CB. Cette manipulation directe du 6301 reste toutefois réservée aux connaisseurs...

L'intérêt de la commande $22, autorisant l'exécution d'une routine du 6301, est plus que discutable à moins que l'on ne désire « planter » le circuit clavier... D'autant que vous ne trouverez nulle part les adresses ROM des routines du 6301 ! A titre informatif, en voici quelques-unes, même si leur utilisation est peu recommandable !

$F000 Reset froid (annule l'horloge).
$F00B Reset chaud (conserve l'horloge).
$F9D1 exécution de la commande $11.
$F9DD exécution de la commande $13.
$F9EF exécution de la commande $14.
$F9F3 exécution de la commande $15.
$F9FA exécution de la commande $16.
$FA3E exécution de la commande $18.
$FA77 exécution de la commande $1A.
$FACE exécution de la commande $1C.
$FAF4 exécution de la commande $8.
$FBC4 exécution de la commande $F.
$FBC9 exécution de la commande $10.
$FBCE exécution de la commande $12.

Les routines d'interruption timer et d'interruption série sont placées respectivement en $FDE9 et $FF23. Encore faut-il souligner que ces adresses sont valables pour la version $F0 de la ROM du 6301, inchangée de l'apparition du 520 ST à janvier 1986 mais susceptible d'être mise à jour.

5.3. Programmation du synthétiseur de son

Les caractéristiques matérielles du circuit synthétiseur de son ont été présentées au chapitre 1 (1.1.3). La description des différents registres, si elle constitue le point de départ obligé pour toute programmation de sons, nécessite toutefois d'être illustrée d'exemples, raison de ce sous-chapitre.

La pré-version du Basic, diffusée à l'heure où sont écrites ces lignes, permet d'offrir un aperçu des possibilités sonores du 520 ST. Le court programme suivant exécute la gamme sur 8 octaves successifs :

```
20 for i = 1 to 8
30 for j = 1 to 12
40 sound 1,15,j,i,10
50 next j
60 next i
70 wave 0
```

Si ce programme ne produit aucun son, il vous est conseillé de régler la hauteur du son sur le moniteur ! Si vous estimez que le son pourrait être plus mélodieux, ajoutez au programme précédent la ligne :

```
10 wave 1,1,9,5000,1
```

Les instructions Basic SOUND et WAVE (son et enveloppe en français) fonctionnent avec les paramètres suivants :

SOUND <adresse> ou <adresse> est l'adresse d'une suite de commandes envoyées au circuit son.

SOUND suivi de 2 à 5 paramètres avec les significations :
- 1er paramètre : numéro du canal (1 à 4)
- 2e paramètre : volume du son (0 à 15)
- 3e paramètre : note (1 à 12)
- 4e paramètre : octave (1 à 8)
- 5e paramètre : temporisation après le son (0 à 65535).

WAVE suivi de 1 à 5 paramètres avec les significations :
- 1er paramètre : nombre compris entre 0 et 63
  - bit 0 : à 1 valide le canal A
  - bit 1 : à 1 valide le canal B
  - bit 2 : à 1 valide le canal C
  - bit 3 : à 1 mixte le canal de bruit et A
  - bit 5 : à 1 mixte le canal de bruit et B
  - bit 5 : à 1 mixte le canal de bruit et C
- 2e paramètre : nombre compris entre 0 et 7
  - bit 0 : à 1 valide l'enveloppe pour A
  - bit 1 : à 1 valide l'enveloppe pour B
  - bit 2 : à 1 valide l'enveloppe pour C

3e paramètre : type de l'enveloppe (0 à 15)
4e paramètre : période de l'enveloppe (0 à 65535)
5e paramètre : temporisation après commande WAVE.

Les paramètres de WAVE correspondent au registre 6 pour le premier, aux bits 4 des registres 8, 9 et $A pour le deuxième, au registre $D pour le troisième, aux registres $B et $C pour le quatrième. La temporisation, en 5e paramètre des commandes SOUND et WAVE correspond à l'envoi d'un certain nombre de demandes de délai (commande $FF de Trap 14, fonction $20). La temporisation est exprimée en cinquantièmes de seconde. Dans la commande SOUND, le premier paramètre correspond au numéro de canal (1 = A, 2 = B, 3 = C) ou au canal de bruit (4). Dans ce dernier cas, le troisième paramètre n'est pas pris en compte et le cinquième désigne la période (0 à 15), correspondant au registre 6 du circuit son.

La note, stockée comme troisième paramètre de SOUND, correspond à un ton de la gamme diatonique majeure (1 = DO, 2 = DO dièse, 3 = RE, 4 = MI bémol, 5 = MI, 6 = FA, 7 = FA dièse, 8 = SOL, 9 = LA bémol, 10 = LA, 11 = Si bémol et 12 =SI). Les octaves 1 à 8 correspondent à des fréquences s'échelonnant entre 32.7 Hz et 7 812.5 Hz (donc sur une gamme réduite mais bien suffisante). Le LA international (à 440 Hz) correspond au LA (note 10) de l'octave 4.

On notera que pour envoyer des sons sur plusieurs canaux il est nécessaire d'employer une commande SOUND par canal, un mixage pouvant ensuite être effectué par le biais de la commande WAVE. On relèvera également que la commande WAVE 0 à pour effet de désactiver la sortie son puisqu'elle désactive les trois canaux de tons et les mixages éventuels avec le canal de bruit.

La programmation de sons, bruits ou mélodies, en langage C comme en assembleur, dépend étroitement du but recherché. Si l'on désire produire des sons ou des séquences de sons identiques, la meilleure méthode consiste à stocker ces sons ou ces séquences sous forme de chaînes de mots, chacun des mots comportant le numéro du registre son concerné dans l'octet fort et la valeur à placer dans ce registre dans l'octet faible. Cette chaîne doit se terminer par le mot $FF00 (indicateur de fin de chaîne son). Son adresse peut alors être envoyée comme paramètre de l'instruction Dosound (Trap 14, fonction $20).

Si l'on désire produire des sons divers et peu répétitifs (pour une mélodie, par exemple), il semble préférable de programmer une procédure C dans ce seul but, procédure dont les paramètres d'entrée autoriseront un appel humainement acceptable des différents sons ! On pourra même, si le son est

un élément déterminant du programme, créer plusieurs sous-routines dédiées à la « traduction » de tel ou tel paramètre sonore.

A titre d'exemple, voici une routine en langage C dont les paramètres d'entrée sont la note et l'octave et qui retourne l'entier servant au codage de la période dans les registres correspondants du canal :

```
int
convfreq(note,octave)
int     note;
int     octave;
{
        int   minoct;
        long  varbase;
static long  freqbase[12] = { 4286448, 4541383, 4811391, 5097521,
                              5400560, 5721817, 6061949, 6422397,
                              6804341, 7208960, 7637565, 8091730 };

        if (note <1) note = 1;            /* traitement d'erreurs  */
        if (note >12) note = 12;          /* pouvant être supprimé */
        if (octave <1) octave = 1;        /* s'il ne semble pas    */
        if (octave >8) octave = 8;        /* indispensable         */
        varbase = freqbase [ note - 1 ];
        minoct = 8 - octave;
        varbase = ( varbase >> minoct );
        return ( 128000000/varbase);
}
```

Cette routine a été testée et utilisée dans plusieurs programmes. L'entier retourné doit être séparé en deux octets, l'octet fort étant placé dans le registre 1 et l'octet faible dans le registre 0 pour le canal A (registres 3 et 2 pour le canal B, 5 et 4 pour le canal C). Le tableau freqbase[] correspond aux dividendes pour les notes DO à SI du 8e octave. Rappelons que la fréquence d'un son est égale à : 125 000/ton, le ton étant le contenu placé dans les registres, en l'occurrence le paramètre de retour de la fonction convfreq().

La fréquence de l'enveloppe, correspondant aux registres $B et $C de la puce son (octet fort en $C et octet faible en $B), est fournie par :

valeur en Hz = 7 812.5 / (contenu des registres $B et $C)

Il est ainsi possible de faire varier la fréquence de l'enveloppe entre 7812.5 Hz et 0.119 Hz, valeurs correspondant à des périodes de 128 microsecondes à 8.39 secondes. La routine C suivante propose un exemple de stockage de valeur dans les registres de période d'enveloppe $B et $C. Le paramètre d'appel « chaîneson » correspond à un pointeur de chaîne, chaîne dans laquelle sont stockées les commandes à envoyer au circuit son, tandis que le paramètre d'appel période désigne la durée désirée de la période d'enveloppe en millisecondes, durée de base suffisante pour la plupart des enveloppes.

```
stocper_env(chaineson, période)
long  chaineson;
int   période;
{
        int octhaut,octbas;

        if (période > 8388) période = 8388;
        période = (période * 7.8125);
        octhaut = (période >> 8);
        octbas  = période - (256 * octhaut);
        octhaut += 0x0C00;
        octbas  += 0x0B00;
        ( *((int *)(chaineson)) = octhaut);
        ( *((int *)(chaineson + 2)) = octbas);
        chaineson += 4;
}
```

La programmation de sons en C comme en assembleur passe par l'utilisation de la fonction Dosound (Trap 14, $20), dont l'unique paramètre d'appel est un pointeur de chaîne, chaîne dans laquelle sont stockés successivement les numéros de registres et les valeurs à y placer. Dans le TOS, la fonction Dosound est traitée de façon fort brève : le pointeur de chaîne est stockée dans une variable système que nous nommerons _ptrson et une autre variable système (octet que nous nommerons _timerson) est réinitialisée à 0. Au cas où le pointeur de chaîne fourni en appel est négatif, la fonction Dosound retourne le contenu courant de _ptrson.

A ce stade, aucun son n'est produit ni même pris en compte ! Tout commence lors de l'interruption horloge système suivante (fréquence : toutes les 20 ms). La variable _ptrson est testée. Si elle est nulle (pas de son en attente), fin de la routine. Sinon, la variable _timerson est testée. Si elle est positive (temporisation), elle est décrémentée de 1 et fin de la routine. Si elle est nulle, le premier octet de la chaîne son est pris en compte. Plusieurs cas peuvent alors se présenter :

1) Cet octet est positif, il est alors considéré comme un numéro de regisre, avec masquage des seuls bits concernés. L'octet suivant, donnée à placer dans le registre, est envoyé dans celui-ci par écriture de $FF8802 (cf. Annexe 3).
2) Cet octet est compris entre $82 et $FF. Il initialise une temporisation. L'octet suivant est saisi et placé dans _timerson, il produit une temporisation égale à : 20 ms * valeur de l'octet. Si l'octet est nul (temporisation nulle), la variable _ptrson est remise à 0 (fin de chaîne).
3) Cet octet est égal à $80. L'octet suivant est stocké dans une variable système de réserve que nous nommerons _réserve.
4) Cet octet est égal à $81. Il s'agit alors d'une commande *à 3 arguments*. Le premier argument est le numéro du registre à charger avec la valeur _réserve précédemment placée par la commande $80. Le second argument est une valeur d'incrémentation (valeur qui sera ajoutée au premier argument). Le

troisième argument est une valeur finale. On retrouve ici, via la variable _réserve, les trois paramètres de base d'une boucle. Cette boucle envoie dans le registre pointé par le premier argument la valeur en _réserve + le deuxième argument, puis la valeur en _réserve + 2 * (deuxième argument), etc... jusqu'à la valeur finale placée en troisième argument.

Une autre fonction système permet de programmer des sons : la fonction SIC de Trap 14. Malheureusement cette fonction n'a pas été enregistrée dans les bibliothèques C et n'est utilisable qu'en assembleur. Elle permet de lire ou écrire dans un registre du circuit son. Le premier paramètre est la donnée à écrire et son second paramètre entier est le numéro du registre (0 à $F). En écriture, le registre doit avoir le bit 7 à 1 (drapeau d'écriture) ; en lecture, ce bit doit être nul.

Cette instruction a le grand intérêt de permettre une utilisation intensive du synthétiseur de son puisque les données sont placées directement dans les registres alors qu'une seule donnée est placée dans un seul registre toutes les 20 ms avec l'instruction Dosound. Mais, d'une part elle n'est actuellement utilisable qu'en assembleur, d'autre part elle réclame un grand soin, toute écriture dans le registre $E (gestion des drives et RS 232) pouvant avoir des conséquences catastrophiques. De plus, aucun contrôle n'est effectué sur les données placées (rappelons par exemple que les bits 6 et 7 du registre servant à fixer la direction des ports A et B et ne doivent pas être modifiés !)

5.4. Programmation des entrées-sorties

Si l'on englobe dans les entrées-sorties la gestion des lectures et écritures sur disque, ce qui semble logique, la programmation des entrées-sorties constitue la majeure partie de la plupart des programmes. Qui ne connaît le sempiternel print « bonjour » du Basic et son homologue printf (« hello/n ») du langage C (eh oui ! le C n'est guère francisé...), archétype de l'entrée-sortie élémentaire (mais bien utile !).

D'ailleurs, si le RAM disque présenté au chapitre 3 exploite peu les entrées-sorties, la calculatrice fournie en accessoire et la plupart des exemples de cet ouvrage usent abondamment des sorties écran. Le GEM lui-même n'est qu'un système presque entièrement dédié aux entrées et sorties graphiques !

Aussi nombre d'instructions C et GEM sont-elles consacrées à cet aspect de la programmation. Si la normalisation ISO qui s'annonce pour le langage C reste un peu l'arlésienne, il demeure que des instructions comme

« printf » ou « scanf », de par leur volonté hégémonique, n'exploitent qu'assez peu les possiblités graphiques d'une machine comme le ST. Et si le GEM autorise une gestion plus souple et mieux appropriée (le contraire serait surprenant !), certaines fonctions système gardent leurs spécificités.

L'étude précédente des périphériques clavier et synthétiseur a montré comment il était possible de dépasser les structures parfois contraignantes du GEM. D'autres horizons sont ouverts par la programmation de l'interface série, dont il a été superficiellement traité au chapitre 1, de l'interface MIDI ou parallèle. Les programmeurs apprécieront par exemple la fonction $24 du Trap 14, laquelle autorise des copies d'écran partielles et personnalisées (par le biais de la table de conversion des couleurs)...

D'une façon générale toute programmation fine des entrées-sorties devra passer sous les fourches caudines du Trap 14, véritable mine autorisant une exploitation complète des potentialités du 520 ST. On ne saurait donc trop recommander l'étude attentive de ces fonctions système ! Le programmeur en assembleur pourra aussi user de la ligne « A », et prochainement de la ligne « F » (cf. chapitre 3), lesquelles contribuent à réduire considérablement le code objet et apportent rapidité et confort dans la programmation et l'exécution.

Le lecteur pourra s'étonner du manque d'informations sur la gestion des disquettes et du disque dur mais, d'une part le disque dur n'a pu être analysé par l'auteur avant la remise du manuscrit, d'autre part ce sujet mériterait à lui seul un ouvrage, à paraître peut-être... Plusieurs fonctions système (toujours Trap 14 !) permettent d'exécuter la lecture, l'écriture ou le formatage de secteur(s) ou de piste. Ces fonctions sont détaillées en Annexe 4 et nous prions le lecteur intéressé de s'y reporter.

Les adresses système et spécialement les adresses physiquement liées à des périphériques d'entrée-sortie fournissent un second filon, presqu'inépuisable, pour la programmation. Rappelons toutefois qu'il est indispensable de se trouver en mode SUPERVISEUR pour lire et écrire des variables système. Aussi est-il parfois plus économique d'appeler par exemple la fonction Logbase (Trap 14, fonction 2) pour connaître l'adresse de base de la mémoire d'écran plutôt que de passer en mode superviseur, de demander le contenu de la variable système $44E.L, puis de repasser en mode utilisateur.

Pour des lectures ou écritures multiples de variables système, une possibilité intéressante est offert par Trap 14, fonction $26, qui autorise l'exécution d'une de vos routines en mode Trap. Il suffit donc de passer en paramètre d'appel l'adresse de votre routine de lecture ou écriture de variables système, puis d'appeler Trap 14, fonction $26. Votre routine se terminera

par un RTS comme d'ordinaire car elle est appelée par le système à l'aide d'un jsr.

Toutes les informations et tous les exemples fournis dans ce chapitre ont été vérifiés et testés. A priori leur utilisation ne devrait poser aucune difficulté. Bien que les concepteurs du TOS garantissent la perennité des fonctions et variables système actuellement utilisées, il n'est toutefois pas impossible que certaines modifications soient effectuées, spécialement sur les variables d'adresses supérieures à $4B2 ou les fonctions Trap 14 de numéro supérieur à $23. Même si ces modifications sont improbables, toute utilisation de ces fonctions ou variables dans un programme devrait donc faire l'objet de tests préalables.

ANNEXE 1.
FONCTIONS C STANDARD

Les fonctions suivantes appartiennent à la bibliothèque standard du C. Leur utilisation dans un programme suppose l'édition de liens avec « GEM-LIB » et, bien souvent, « LIBF ». Seules les fonctions implémentées sur le compilateur de Digital Research font l'objet de cette nomenclature. On notera à ce propos que certaines fonctions appartenant à la liste des fonctions disponibles sur ce compilateur ne sont pas implémentées sur les 520 ST et donc non étudiées ici. Les fonctions sont classées par ordre alphabétique, le mode d'appel de la fonction ne comportant pas la longueur des paramètres, cette dernière étant indiquée à la suite de chacun des paramètres.

abort. abort (code)

Provoque l'arrêt du programme avec code d'erreur. Utilisé pour « déboguer » un programme à l'aide de SID.

int code : code placé dans le registre D0 avant l'appel.

abs. ret = abs (valeur)

Fournit la valeur absolue de l'argument entier (cf. fabs). Définie comme macro dans « stdio.h » sous la forme : ((x) < 0 ? − (x) : (x)). Simple, non ?

int valeur : argument en entrée.
int ret : valeur absolue de l'argument en entrée.

access. ret = access (nom, mode)

Recherche la possibilité d'accès à un fichier. Pour un nom et un type de fichier donné, est-il possible d'y accéder ?

char *nom : pointe la chaîne du nom de fichier.

int type : 4 pour vérifier l'accès en lecture, 2 en écriture,
1 pour vérifier qu'il s'agit d'un fichier exécutable.
int ret : 0 si l'accès est possible sous ce mode, – 1 sinon.

atoi, atof, atol. valeur = atoi (chaîne)

Conversion d'une chaîne ASCII en entier (atoi), flottant (atof) ou entier long (atol). ATTENTION, le débordement n'est pas détecté.

char *chaîne : chaîne ASCII devant se terminer par un caractère nul.
valeur : int si atoi, float si atof, long si atol.

brk, sbrk ret = brk (adresse) début = sbrk (taille)

Gèrent l'espace réservé à l'application. La fonction brk fixe la limite haute de la zone réservée. La fonction sbrk permet d'étendre cette zone. Dans le compilateur de D.R., « gemstart » commence par l'appel de brk.

char *adresse : adresse désirée du haut de l'application.
int taille : nombre d'octets à ajouter à la zone réservée.
char *début : adresse de départ de la nouvelle zone réservée (0 si la fonction sbrk n'a pu être exécutée).

calloc, malloc, realloc, free, adresse = malloc (taille)
adresse = calloc (nombre, taille), realloc (adresse, taille) free (adresse)

La fonction malloc alloue une zone de mémoire de « taille » octets. La fonction free libère cette zone allouée. La fonction calloc alloue « nombre » fois la « taille » précisée (allocation pour tableaux ou structures). La fonction realloc change la taille d'une zone allouée. ATTENTION, la taille de la zone est un entier, donc inférieure à 65 536 octets.

char *adresse : adresse de la zone à allouer ou ré-allouer.
int taille : nombre d'octets à allouer.
int nombre : nombre d'éléments pour « calloc ».

ceil ret = ceil (valeur)

Fournit l'entier arrondi par excès d'une valeur flottante. Cet entier est stocké *sous forme de nombre flottant* (cf. floor).

float valeur, ret : argument et résultat flottants.

chmod, chown. ret = chmod (nom, mode) ret = chown (nom, prop, groupe)

Primitives UNIX non prises en compte par le compilateur C de D. R.

close. ret = close (id_fich)

Fermeture d'un fichier dont id_fich est l'identificateur (voir creat).

int fd, ret : identificateur et retour (0 si OK, – 1 sinon).

cos, sin. ret = cos (valeur ret = sinus (valeur)

Fournit le cosinus (ou le sinus) d'un flottant exprimé en radians.

float valeur, ret : argument et résultat.

creat, creata, creatb. id_fich = creat (nom, mode)

Ajoute un fichier à un catalogue disquette. Les fonctions creat et creata sont identiques et créent un fichier ASCII. La fonction creatb crée un fichier binaire. Voir open, close, read et write.

char *nom :	pointe la chaîne de stockage du nom de fichier.
int mode :	non pris en compte (0 par exemple).
int id_fich :	identificateur du fichier (A CONSERVER !).

etoa, ftoa. ret = etoa (val, tamp, prec) ret = ftoa (val, tamp, prec)

Conversion d'un nombre flottant en chaîne ASCII. Fonctions identiques.

float val :	valeur flottante à convertir.
char *tamp :	adresse du tampon de stockage de la chaîne résultat.
int prec :	précision (nombre de chiffre après la virgule, 0 à 5).
char *ret :	adresse de la chaîne résultat (identique à tamp).

exit,_exit. exit (code) _exit(code)

La fonction exit provoque une sortie du programme avec fermeture des fichiers ouverts et désallocation des zones de mémoire allouées. La fonction _exit est identique mais sans fermeture ou désallocation.

int code : valeur retournée après sortie du programme.

exp. ret = exp (val)

Fournit l'exponentielle d'une valeur flottante.

float val, ret : argument et résultat *flottants.*

fabs. ret = fabs (val)

Fournit la valeur absolue d'une valeur flottante (cf. abs).

float val, ret : argument et résultat *flottants.*

fclose, fflush. ret = fclose (canal) ret = fflush (canal)

Respectivement fermeture et vidage des tampons d'un canal (cf. fopen).

FILE *canal : structure de données sur le canal (voir fopen).
int ret : 0 si OK, − 1 si l'adresse de canal est erronée.

feof, ferror, clearerr, fileno. ret = feof (canal) ret = ferror (canal) clearerr (canal) descr = fileno (canal)

Manipulation directe de canal. «feof » effectue un test de fin de fichier. « ferror » effectue un test d'erreur et « clearerr » supprime cette erreur. « fileno » renvoie le descripteur du fichier associé au canal.

FILE *canal : structure de données sur le canal (voir fopen).
int ret : 0 pour « feof » si fichier non terminé.
int descr : descripteur de fichier.

floor. ret = floor (valeur)

Retourne l'entier arrondi par défaut d'un flottant (cf. ceil).

float valeur, ret : argument et résultat *flottants.*

fmod. ret = fmod (a, b)

Fournit le reste de la division de a par b (le « modulo »).

float a,b, ret : a est le diviseur, b est le dividende, ret le reste.

fopen, freopen, fdopen. canal = fopen (nom, accès) canal = fdopen (descr, access)

Association d'un canal avec un fichier. « fopen » et « fopena » associent un fichier ASCII à un canal, « fopen » associe un fichier binaire. « freopen » et « freopena » substituent un nouveau fichier ASCII à un canal ouvert (« fopenb » pour un fichier binaire). « fdopen » associe un fichier déjà ouvert à un canal.

La structure FILE est définie ainsi :

int _fd descripteur de fichier.
int _flag drapeau d'erreur ou de fin de fichier.
char *_base pointe la base du tampon du fichier.
char *_ptr pointe la position courante dans le tampon.
int _count nombre de caractères à lire ou écrits.

FILE *canal : structure de données sur le canal.
char *nom : pointe la chaîne du nom de fichier.

char *access : pointe la chaîne « r » si lecture, « w » si écriture, « a » si ajout à un fichier.
int descr : descripteur de fichier (voir fileno).

fread, fwrite. fread (tamp, taille, nombre, canal)

Lecture ou écriture d'octets d'un canal vers la mémoire.

char *tamp : pointe le début du tampon en mémoire.
int taille : nombre d'octets dans chaque item.
int nombre : nombre d'items à lire ou écrire.
FILE *canal : structure de données du canal (voir fopen).

fseek, ftell, rewind. ret = fseek (canal, dep, type)
dep = ftell (canal) ret = rewind (canal)

Positionnement du pointeur de canal. « fseek » positionne le pointeur, « ftell » fournit la position et « rewind » place le pointeur en début de canal.

FILE *canal : structure de données du canal (voir fopen).
long dep : nombre d'octets du déplacement du pointeur.
int type : type de déplacement (0: à partir du début du fichier, 1 : à partir de la position courante, 2 : à partir de fin).
int ret : 0 si OK, −1 sinon.

getc, getchar, fgetc, getw, getl. carac = getc (canal)

Lecture de caractère à partir d'un canal (getc et fgetc : lecture d'un seul caractère [getchar à partir de la console] ; getw : lecture d'un entier 16 bits ; getl : lecture d'un long mot 32 bits).

FILE *canal : structure de données du canal (voir fopen).
carac : char si getc ou fgetc, int si getw, long si getl.

getpass. mot = getpass (chaîne)

Lit un « mot de passe » entré au clavier. « chaîne » est affiché et un curseur apparaît. Les caractères entrés ne sont pas affichés. Huit caractères au maximum peuvent être entrés.

char *chaîne : pointe la chaîne affichée.
char *mot : pointe le mot lu au clavier.

gets, fgets. adresse = gets(s) adresse = fgets (s, n, canal)

Lecture de chaînes à partir d'un canal. « fgets » lit une chaîne terminée par un retour de chariot, « gets » supprime le code de chariot et ne lit que sur le canal clavier.

char *adresse : pointe l'adresse de stockage de la chaîne.
char *s : pointe le tampon de stockage.
int n : nombre maximum d'octets à lire.
FILE *canal : structure de données sur le canal (voir fopen).

index, rindex. adresse = index (chaîne, code)

Recherche d'un caractère dans une chaîne (index fournit la première occurrence et rindex la dernière).

char *chaîne : pointe la chaîne ou rechercher le caractère.
char code : code du caractère à rechercher.
char *adresse : adresse du caractère recherché.

isatty. ret = isatty (id_fich)

Teste si un fichier est lié au clavier ou non.

int id_fich : identificateur de fichier (voir open).
int ret : 1 si fichier lié au clavier, 0 sinon.

log. ret = log (valeur)

Fournit le logarithme néperien d'une valeur flottante.

float valeur, ret : argument et résultat *flottants*.

lseek, tell. ret = lseek (id_fich, dep, type) ret = tell (id_fich)

« lseek » positionne et « tell » fournit la position d'un pointeur de fichier (correspondant à fseek et ftell pour un canal).

int id_fich : identificateur de fichier (voir open).
long dep : nombre d'octets du déplacement de la position.
int type : 0, 1 ou 2 (voir fseek).
lont ret : position absolue si ftell, −1 si erreur.

mktemp. chaîne = mktemp (chaîne)

Création d'un nom de fichier temporaire. A l'appel, « chaîne » pointe sur une chaîne se terminant par XXXXXX. Au retour ces caractères sont recouverts par le nom temporaire.

char *chaîne : pointe la chaîne calibrée puis modifiée.

open, opena, openb. id_fich = open (nom, mode)

Ouverture d'un fichier ASCII pour fopen et fopena, binaire pour fopenb.

char *nom : pointe la chaîne de stockage du nom de fichier.

int mode : 0 en lecture, 1 en écriture, 2 en lecture/écriture.
int id_fich : identificateur de fichier (A CONSERVER !).

perror. erreur = perror (chaîne)

Affiche un message d'erreur décrivant la dernière erreur rencontrée.

char *chaîne : pointe la chaîne à afficher.
int erreur : numéro de l'erreur lors de l'appel.

pow. ret = pow(x,y)

Fournit la valeur de x à la puissance y.

float x, y, ret : arguments et résultats *flottants.*

printf, fprintf, sprintf printf (chaîne, arg1, arg2,...)

fprintf (canal, chaîne, arg1, arg2,...) sprintf (canal, chaîne, arg1,...)

"printf" affiche une chaîne formatée, "fprintf" sort une chaîne sur un canal donné, "sprintf" place une chaîne en mémoire. Le symbole % dans la chaîne formatée indique un opérateur, le caractère le suivant indiquant le type de l'opérateur (d = nombre décimal en ACSII, o = nombre binaire en octal ASCII x = nombre binaire en hexadécimal ASCII, c = nombre en simple caractère, s = nombre en adresse de chaîne, u = nombre décimal non signé en ASCII).

char *chaîne : chaîne formatée avec opérateurs éventuels.
arg1, arg2... : arguments de conversion pour opérateurs.
FILE *canal : structure de données décrivant le canal (cf. fopen).

putc, putchar, fputc, putw, putl. ret = putc (car,canal)

Sortie d'un caractère à l'écran (putchar), sur un canal (putc, fputc).
Sortie d'un mot de 16 bits avec putw, de 32 bits avec putl.

char car : caractère à sortir.
FILE *canal: structure de données décrivant le canal (cf. fopen).
int ret : caractère placé (LONG si putl), −1 si erreur.

puts, fputs. ret = puts (chaîne) ret = fputs (chaîne,canal)

Sortie d'une chaîne terminée par un caractère nul à l'écran avec retour de chariot (puts) ou sur un canal sans retour à la ligne (fputs).

char *chaîne : pointe la chaîne à afficher.
FILE *canal : structure de données décrivant le canal (cf. fopen).
int ret : dernier caractère sorti, −1 si erreur.

qsort. qsort (base, nombre, taille, compare).

Tri rapide de « nombre » éléments comportant « taille » octets et stockés à partir de « base », « compare » est l'adresse d'une *routine* de comparaison du type "ret = compare (a, b)", renvoyant − 1 si a < b, 0 si a = b, 1 si a > b.

char *base : adresse de départ du tableau à trier.
int nombre : nombre d'éléments à trier.
int taille : taille de chaque élément.
int compare() : adresse de la routine de comparaison.

rand, srand. alea = rand() alea = srand (semence)

Fournit un nombre pseudo-aléatoire quelconque (rand) ou déduit d'une « semence » (srand).

int semence, ret : argument éventuel et résultat entiers.

read. ret = read (id_fich, tampon, nombre)

Lecture d'octets dans un fichier (voir open et creat).

int id_fich : identificateur de fichier (voir open).
char *tampon : pointe le tampon où stocker les données lues.
int nombre : nombre d'octets à lire.
int ret : nombre d'octets lus, − 1 si erreur.

scanf, fscanf, sscanf. nombre = scanf (chaîne, arg1, arg2, ...)

nombre = fscanf (canal, chaîne, arg1, ...) nombre =sscanf (ch1, chaîne, arg1, ...)

Conversion de chaîne entrée au clavier (scanf), sur un canal (fscanf) ou à partir d'une chaîne (scanf). Voir « printf » pour les formats.

char *chaîne : chaîne formatée (voir printf).
char *ch1 : chaîne d'origine pour sscanf.
FILE *canal : structure de données décrivant le canal (cf. fopen).
arg1, arg2 : données pour opérateurs de formatage.
int nombre : nombre de données converties.

setjmp, longjmp. ret = setjmp (env) longjmp (env, ret)

Autorise un GOTO externe à une procédure, « env. » doit pointer un tampon de 50 octets, « ret » fournit un identificateur. A partir de ces deux paramètres, la fonction « longjmp » permet un saut du programme *à l'instruction suivant setjmp* avec des registres identiques à ceux de l'appel de setjmp.

char *env : adresse d'un tampon de 50 octets minimum.
int ret : identificateur de saut

signal. ret = signal (numéro, fonc)

Permet de faire correspondre une routine C à une exception 68000 (indiquée par « numéro » : 4 = illégal, 5 = trace, 6 = traps différents de 2 et 3, 8 = division par 0, TRAPV et CHECK, 10 = erreur de bus).

int signal : numéro de l'exception (voir ci-dessus).
int fonc() : adresse de la *routine C.*
int ret : 0 si OK, − 1 si numéro non autorisé.

sqrt. ret = sqrt (valeur)

Fournit la racine carrée d'un nombre flottant.

float valeur, ret : argument et résultats *flottants.*

strcat. strncat. ret = strcat (S1, S2) ret = strcat (s1, s2, n)

Concaténation de chaînes. « strncat » n'ajoute à s1 que n caractères de s2.

char *s1, *s2 : pointent les chaînes à concaténer.
int n : nombre d'octets à ajouter pour stncat.
char *ret : pointe le début de la chaîne concaténée.

strcmp, stncmp. val = strcmp (s1, s2) val = strncmp (s1, s2, n)

Comparaison de chaînes, sur n caractères pour « strncmp ».

int n : nombre de caractères à comparer si strncmp.
int ret : − 1 si s1 < s2, 0 si s1 = s2, 1 si s1 > s2.

strcpy, strncpy. ret = strcpy (s1, s2) ret = strncpy (s1, s2, n)

Copie d'une chaîne, sur n caractèress si « strncpy ».

char *s1, *s2 : pointent la chaîne à copier et la chaîne copiée.
int n : nombre de caractères à copier si strncpy.
char *ret : pointe le début de la chaîne s1.

strlen. longueur = strlen(s)

Fournit la longueur d'une chaîne.

char *s : pointe la chaîne.
int longueur : longueur de la chaîne.

swab. ret = swab (source, dest, nombre)

Copie une zone de mémoire dans une autre avec inversion des octets faibles et forts (utile pour conversion de données 8086 en 68000).

char *source : pointe la zone de mémoire à copier.
char *dest : pointe la zone de mémoire où copier.
int nombre : nombre *PAIR* d'octets à copier.

tan, atan. ret = tan (valeur) ret = atan (valeur)

Fournissent la tangente ou l'arc tangente d'un nombre flottant.

float valeur, ret : argument et résultat *flottants.*

ttyname. nom = ttyname (id_fich)

Retourne un pointeur sur la chaîne « CON: » si le fichier est affecté au clavier, une chaîne nulle sinon.

int id_fiche : identificateur de fichier (voir open).
char *nom : pointe la chaîne résultat (« CON: » ou nulle).

ungetc. ret = ungetc (car, canal)

Place un caractère dans le tampon *d'entrée* d'un canal. Ce caractère sera donc disponible au prochain appel « getc ».

char car : caractère à forcer dans le tampon d'entrée.
FILE *canal : structure de données décrivant le canal (cf. fopen).
int ret : le caractère si ok, – 1 sinon.

unlink. ret = unlink (nom)

Effacement d'un nom de fichier si celui-ci existe et n'est pas ouvert.
char *nom : pointe la chaîne du nom du fichier.
int ret : 0 si ok, – 1 si erreur.

write. ret = write (id_fich, tampon, nombre)

Écriture de données dans un fichier ouvert. L'écriture a lieu là où est positionné le pointeur de fichier (voir lseek). Attention, l'écriture dans le fichier n'est certaine qu'une fois celui-ci fermé.

int id_fich : identificateur de fichier (voir open).

char *tampon : pointe le tampon où sont stockées les données à écrire.
int nombre : nombre d'octets à écrire.
int ret : nombre d'octets écrits, – 1 si erreur.

ANNEXE 2
LES FONCTIONS GEM

Les fonctions VDI

Avant de fournir la liste exhaustive des fonctions VDI et les paramètres particuliers de chacune d'elles, quelques notions générales doivent être explicitées.

Les fonctions VDI, utilisent, pour interagir avec le programme appelant, cinq zones d'arguments dont chaque élément est un mot (deux octets, soit un entier pour un microprocesseur 16 bits). Ces cinq zones sont reconnues par l'éditeur de liens sous les noms de tableaux suivants :

contrl [] : zone de contrôle de la fonction
intin [] : zone des paramètres d'entrée
ptsin [] : zone des coordonnées x et y de points en entrée
intout [] : zone des paramètres de sortie
ptsout[] : zone des coordonnées x et y de points en sortie

Les valeurs et tailles de ces zones varient selon la fonction, exception faite de la zone de contrôle, dont les sept premiers éléments référent, pour chaque appel, au même contenu :

Paramètres en entrée :

contrl [0] : Numéro de la fonction.
contrl [1] : Nombre de points en entrée. Comme chaque point nécessite deux coordonnées (x et y), contrl [1] est égal à la moitié du nombre d'arguments de ptsin.
Mis à 0 si aucun argument en ptsin.
contrl [3] : longueur du tableau intin.
Mis à 0 si longueur nulle.

contrl [5] : Numéro de la sous-fonction, Nécessaire lorsque le numéro de fonction n'est pas suffisamment explicite. (ex. : fonction 5, fonction 11).

Paramètres en sortie

contrl [2] : Nombre de points en sortie. Voir contrl[1] mais relativement à ptsout.

contrl [4] : Longueur du tableau intout.

Paramètre d'entrée ou de sortie, selon la fonction

contrl [6] : Numéro de l'identificateur VDI.

REMARQUE GÉNÉRALE :

la liste des fonctions VDI a été classée par thème, choix qui a paru judicieux à l'auteur au regard de son expérience de programmation.

Pour cette raison, certaines caractéristiques communes à différents titres de fonction (titres qui ne doivent rien au hasard) peuvent ne pas apparaître et priver ainsi le lecteur de certains moyens mnémotechniques susceptibles de le guider.

La nomenclature obéit aux principes suivants :

1) Toute fonction VDI commence par la lettre v.
2) la lettre s (pour SET) placée en seconde position indique que la fonction fournit un ou des paramètres au VDI.
3) la lettre q (pour INQUIRE), elle aussi placée comme deuxième lettre, est utilisée dans les fonctions dont le rôle spécifique est de vous renvoyer des paramètres.
4) Après un q ou un s, la troisième lettre placée avant le trait de soulignement (_) sert souvent à déterminer le rôle de la fonction : f pour FILL (remplissage), t pour TEXT, l pour LINE (ligne), m pour MARKER (marqueur), etc...

Fonctions générales

v_opnwk fonction 1

Ouvre une station de travail. Mêmes arguments que v_opnvwk.

Le VDI interdit l'ouverture de plusieurs stations de travail. Le Bureau GEM en ayant ouvert une lors de son chargement, cette fonction n'est guère utilisable que dans un bootstrap ! (cf. chapitre 3 : les interruptions).

Pour lancer votre application du Bureau, il faut donc utiliser la fonction v_opnvwk (voir plus loin).

v_clswk fonction 2

Ferme une station de travail. Utilise les mêmes arguments que v_clsvwk. Même remarque que ci-dessus.

v_opnvwk fonction 100

Ouvre une station de travail virtuelle.

Votre application devient ainsi « propriétaire » de la station de travail et de l'écran et peut initialiser ses paramètres spécifiques sans qu'il y ait conflit avec la station de travail ouverte par le Bureau.

Retourne des informations système et fournit l'identificateur VDI de votre application.

```
int id_vdi, arg_in [11], arg_out [57] ;
v_opnvwk (arg_in, &id_vdi, arg_out) ;
```

Paramètres d'appel :

id_vdi (contrl [6]) : doit contenir en entrée (et en fait contient par défaut) l'identificateur de la station de travail ouverte (soit la valeur 1, identificateur du Bureau GEM).
Au retour, contient l'identificateur VDI de votre application.

arg_in[0] (intin [0]) : identificateur du périphérique (1 = écran)
arg_in[1] (intin [1]) : type de ligne
arg_in[2] (intin [2]) : index de couleur de ligne
arg_in[3] (intin [3]) : type de marqueur
arg_in[4] (intin [4]) : index de couleur de marqueurs
arg_in[5] (intin [5]) : police de caractères (1 = police système)
arg_in[6] (intin [6]) : index de couleur du texte
arg_in[7] (intin [7]) : style intérieur de remplissage
arg_in[8] (intin [8]) : index de style de remplissage
arg_in[9] (intin [9]) : index de couleur de remplissage.
arg_in[10] (intin [10]) : système de coordonnées : par défaut, 2 système RC seul valide

Paramètres de retour :

Est spécifié pour chaque argument la réponse (entre crochets) fournie par le GEM de l'ATARI.

arg_out[0] (intin [0]) : nombre de points adressables dans la largeur de l'écran. [639 en haute et moyenne résolution (+ point 0 = 640)] [319 en basse résolution (+ point 0 = 320)]

arg_out[1] (intout [1]) : nombre de points adressables dans la hauteur de l'écran. [399 en haute résolution (+ point 0 = 400)] 199 en basse et moyenne résolution (+ point 0 = 200)]

arg_out[2] (intout [2]) : système permettant la réduction d'une image à l'échelle. [NON (= 0)]

arg_out[3] (intout [3]) : largeur d'un pixel en microns. [372 en monochrome]

arg_out[4] (intout [4]) : hauteur d'un pixel en microns. [372 en monochrome]

arg_out[5] (intout [5]) : nombre de hauteur de caractères. [3].

arg_out[6] (intout [6]) : nombre de types de ligne. [7]

arg_out[7] (intout [7]) : nombre d'épaisseurs de ligne. [toute épaisseur]

arg_out[8] (intout [8]) : nombre de type de marqueurs. [6]

arg_[9] (intout [9] : nombre de tailles de marqueurs. [8]

arg_out[10] (intout [10]) : nombre de jeux de caractères. [1]

arg_out[11] (intout [11]) : nombre de fonds spéciaux de remplissage. [24]

arg_out[12] (intout [12]) : nombre de fonds hachurés de remplissage. [12]

arg_out[13] (intout [13]) : nombre de couleurs pouvant être affichés en même temps à l'écran. [2 pour H.R., 4 pour M.R., 16 pour B.R.]

arg_out[14] (intout [14]) : nombre de formes pré-définies. [10]

arg_out[15] à arg_out[24] (soit de intout[15] à intout[24]) : numéro des formes supportées (voir fonctions de traçage de formes)

arg_out[25] à arg_out[34] (soit de intout[25] à intout[34]) : attributs associées à chaque forme (voir fonctions traçage)

arg_out[35] (intout [35]) : drapeau couleur [0 si écran mono, 1 si écran couleur]

arg_out[36] (intout [36]) : possibilité d'inclinaison du texte. [OUI (= 1)]

arg_out[37] (intout [37]) : possibilité de remplissage de zone. [OUI (= 1)]

arg_out[38] (intout [38]) : possibilité de division d'un rectangle en cellules indépendantes. [NON (= 0)]

arg_out[39] (intout [39]) : nombre de couleurs possibles dans la palette. [2 si écran mono, 512 si écran couleur]

arg_out[40] (intout [40]) : nombre de périphériques gérant la souris. [2]

arg_out[41] (intout [41]) : périphérique gérant le curseur. [CLAVIER]

arg_out[42] (intout [42]) : périphérique gérant les touches. [CLAVIER]

arg_out[43] (intout [43]) : périphérique gérant les chaînes. [CLAVIER]

arg_out[44] (intout [44]) : type de station de travail. [ENTRÉE-SORTIE]
arg_out[45] (ptsout [0]) : largeur minimale de caractères. [5]
arg_out[46] (ptsout [1]) : hauteur minimale de caractères. [4]
arg_out[47] (ptsout [2]) : largeur maximale de caractères. [7]
arg_out[48] (ptsout [3]) : hauteur maximale de caractères. [14]
arg_out[49] (ptsout [4]) : épaisseur minimale de ligne. [1]
arg_out[50] (ptsout [5]) : 0
arg_out[51] (ptsout [6]) : épaisseur maximale de ligne. [40]
arg_out[52] (ptsout [7]) : 0
arg_out[53] (ptsout [8]) : largeur minimale d'un marqueur. [15]
arg_out[54] (ptsout [9]) : hauteur minimale d'un marqueur. [11]
arg_out[55] (ptsout [10]) : largeur maximale d'un marqueur [120]
arg_out[56] (ptsout [11]) : hauteur maximale d'un marqueur. [88]

vq_extnd fonction 102

Retourne des informations supplémentaires, non fournies par v_opnvwk.

int id_vdi, drapeau, arg_out [57] ;

vq extnd (id_vdi, drapeau, arg_out) ;

Paramètres d'appel

id_vdi (contrl [6]) : identificateur VDI
drapeau (intin [0]) : si 0, retourne les mêmes valeurs que v_opnwk
: si 1, retourne les valeurs qui suivent

Paramètres de retour

arg_out[0] (intout [0]) : type de l'écran.
[Moniteur graphique et alphabétique avec image commune]
arg_out[1] (intout [1]) : nombre de couleurs de fond.
[1 pour écran mono, 511 pour écran couleur]
arg_out[2] (intout [2]) : effets spéciaux sur texte (Cf fonctions texte)
arg_out[3] (intout [3]) : mise à l'échelle possible. [NON (= 0)]
arg_out[4] (intout [4]) : nombre de plans. (Cf. chap. 1.1.2.)
[1 pour H. Resolution, 2 pour M.R., 4 pour B.R.]
arg_out[5] (intout [5]) : table de palettes modifiable. [OUI (= 0)]
arg_out[6] (intout [6]) : vitesse de copie d'écran.
[16 ∧ 2 pixels = 1/1 000 sec]
arg_out[7] (intout [7]) : possibilité de remplissage [OUI (= 1)]
arg_out[8] (intout [8]) : inclinaison de caractères. [Par 90°]
arg_out[9] (intout [9]) : nombre de modes d'écriture. [4]
arg_out[10] (intout [10]) : mode d'interrogation des entrées. [2]

arg_out[11] (intout [11]) : autorise l'alignement de texte. [OUI (= 1)]
arg_out[12] (intout [12]) : encre autorisé. [NON]
arg_out[13] (intout [13]) : [0]
arg_out[14] (intout [14]) : longueur maximale de zone ptsin. [128]
arg_out[15] (intout [15]) : longueur maximale de zone intin. [illimité]
arg_out[16] (intout [16]) : nombre de boutons de souris. [2]
arg_out[17] (intout [17]) : style pour ligne large. [NON]
arg_out[18] (intout [18]) : mode d'écriture pour ligne large. [NON]
arg_out[19] (intout [19]) : réservé, mis à 1.
arg_out[20] à arg_out[44] (soit de intout[20] à intout[44]) : réservé [0]
arg_out[45] (ptsout [0]) : coordonnée X(min) de vs_clip.
arg_out[46] (ptsout [1]) : coordonnée Y(min) de vs_clip.
arg_out[47] (ptsout [2]) : coordonnée X(max) de vs_clip.
arg_out[48] (ptsout [3]) : coordonnée Y(max) de vs_clip.
arg_out[49] à arg_out[56] (soit ptsout[4] à ptsout[11]) : réservé [0]

v_clsvwk fonction 101

Ferme la station de travail virtuelle ouverte par v_opnvwk.

```
int id_vdi ;

v_clsvwk (id_vdi) ;
```

Paramètre d'appel : id_vdi (contrl [6] : identificateur VDI

v_clrwk fonction 3

Efface l'écran avec la couleur de fond d'indice 0 (blanc).

```
int id_vdi ;

v_clrwk (id_vdi) ;
```

Paramètre d'appel : id_vdi (contrl [6]) : identificateur VDI

vs_clip fonction 129

Cette fonction restreint l'affichage (lorsque le drapeau est à 1) à une zone rectangulaire spécifiée. Tout traçage à l'extérieur de ce rectangle ne sera pas pris en compte.

L'appel de cette fonction avec le drapeau mis à 0 (valeur par défaut à l'ouverture d'une station de travail), autorise l'affichage global.

```
int id_vdi, drap_clip, zone_rec[4] ;

vs_clip (id_vdi, drap_clip, zone_rec) ;
```

Paramètres d'appel :

id_vdi (contrl [6] : identificateur VDI
drap_clip (intin [0]) : drapeau mis à 1 ou 0
zone_rec [0](ptsin [0]) : coord. X du coin supérieur du rectangle
zone_rec [1] (ptsin [1]) : coord. Y du coin supérieur du rectangle
zone_rec [2] (ptsin [2]) : coord. X du coin diagonalement opposé du rectangle.
zone_rec [3] (ptsin[3]) : coord. Y du coin diagonalement opposé du rectangle.

vswr_mode fonction 32

Spécifie le mode d'écriture utilisé pour tout affichage ultérieur, tant en ce qui concerne le traçage de ligne, le remplissage, ou l'affichage de texte sur écran graphique. Quatre modes sont possibles :

1 = normal : toute nouvelle écriture est insensible au contenu de l'ancien écran.
2 = transparent : sur-impression sur l'ancien écran
3 = XOR : sur-impression en vidéo inverse sur l'ancien écran
4 = transparent inverse : sous-impression sous l'ancien écran (cf. chap. 3.3. instruction A003).

int id_vdi, mode, sel_mode :

sel_mode = vswr_mode (id_vdi, mode) ;

Paramètres d'appel :

id_vdi (contrl [6]) : identificateur vdi
mode (intin[0]) : mode à sélectionner soit 1, 2, 3 ou 4.
En cas de valeur invalide, le mode 1 sera sélectionné.

Paramètre de retour :

sel_mode (intout[0]) : le mode effectivement sélectionné.

Fonctions de traçage de lignes

vsl_type fonction 15

Cette fonction définit le type de ligne à utiliser pour tout traçage ultérieur. Un type est défini par un entier de seize bits, constituant le masque de la ligne, l'aspect final de cette ligne dépendant, bien sûr, du mode d'écriture (cf. vswr_mode).

Sept types de ligne sont possibles :

TYPE	BIT	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1 = continu		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2 = trait long		1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
3 = point		1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0
4 = trait-point		1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
5 = trait		1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
6 = trait-point-point		1	1	1	1	0	0	0	1	1	0	0	1	1	0	0	0
7 = défini par l'utilisateur		(Cf fonction vsl_udsty)															

```
int id_vdi, type, sel_type ;

sel_type = vsl_type (id_vdi, type) ;
```

Paramètres d'appel

id_vdi (contrl[6]) : identificateur VDI
type (intin[0]) : numéro du type de ligne (1 par défaut)

Paramètre de retour :

sel_type (intout[0]) : type de ligne effectivement sélectionné

vsl_udsty fonction 113

Permet de définir son propre type de ligne en spécifiant le masque de 16 bit qui sera utilisé pour le type de ligne numéro 7 (Cf. vsl_type).

Le bit 0 du masque correspond au premier pixel de la ligne.

Il est à noter que, si on utilise la couleur, certains masques dans certains modes peuvent avoir un effet pervers (cf. chap. 3.3.).

```
int id_vdi, masque ;

vsl_udsty (id_vdi, masque) ;
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI
masque (intin[0]) : masque de ligne

vsl_width fonction 16

Définit l'épaisseur de ligne en pixel pour tout traçage ultérieur. La valeur envoyée doit être impaire, comprise entre 1 et 39.

```
int id_vdi, épais, sel_épais ;

sel_épais = vsl_width (id_vdi, épais) ;
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI
épais (ptsin[0]) : épaisseur de ligne demandée

Paramètre de retour :
sel_épais (ptsout[0]) : épaisseur effectivement sélectionnée

vsl_ends fonction 108

Définit la forme de début et de fin de ligne pour tout traçage ultérieur. Début et fin de ligne peuvent être de styles différents, et choisis parmi les trois styles suivants :

0 = carré (par défaut)
1 = flèche
2 = arrondi

```
int id_vdi, deb_ligne, fin_ligne ;

vsl_ends (id_vdi, deb_ligne, fin ligne) ;
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI
del_ligne (intin[0]) : numéro du style de début de ligne
fin_ligne, (intin[1]) : numéro du style de fin de ligne

v_pline fonction 6

Cette fonction trace une série de lignes (64 au maximum), en se référant aux points des coordonnées X et Y du tableau d'entrée. La ligne sera tracée en utilisant les attributs par défaut ou ceux que vous avez spécifiés.

Le tableau d'entrée doit contenir au minimum 2 coordonnées (4 éléments) ; pour tracer un point, il suffira qu'elles soient identiques.

```
int id_vdi, nb, coord_xy [2 * nb_ligne] ;

v_pline (id_vdi, nb, coord_xy) ;
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI
nb (contrl[1]) : nombre de lignes à tracer
coord_xy[0] (ptsin[0]) : coord. X du premier point
coord_xy[1] (ptsin[1]) : coord. Y du premier point
...
coord_xy[2nb − 2] (ptsin[2nb − 2]) : coord. X du dernier point
coord_xy[2nb − 1] (ptsin[2nb − 1]) : coord. Y du dernier point

vql_attributes fonction 35

Cette fonction retourne les attributs qui affectent le traçage de lignes.

int id_vdi, attrib[4] ;

vql_attributes (id_vdi, attrib) ;

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :

attrib[0] (intout[0]) : type de ligne (Cf vsl_type)
attrib[1] (intout[1]) : couleur de ligne (Cf vsl_color)
attrib[2] (intout[2]) : mode d'écriture (Cf vswr_mode)
attrib[3]) intout [3]) : épaisseur de ligne (Cf vsl_width)

Fonctions de traçage de formes

Ces fonctions permettent d'appeler des formes prédéfinies par le système, avec un minimum de paramètres. La plupart ayant un numéro de fonction identique, c'est donc le code de la sous-fonction (en contrl [5]) qui va les différencier.

Les fonctions utilisant les mêmes paramètres d'appel ont été regroupées, en soulignant leurs quelques différences.

vr_recfl fonction 114
v_bar fonction 11, sous-fonction 1
vr_box fonction 11, sous-fonction 8
vrf_box fonction 11, sous-fonction 9

Ces quatre fonctions dessinent un rectangle dont on spécifie les coordonnées X et Y de deux coins opposés.

vr_recfl dessine un rectangle sans contour.

v_bar dessine un rectangle (contours possibles).

vrf_box dessine un rectangle aux angles arrondis.

Pour ces trois fonctions, les attributs référant au remplissage de zone (cf. vqf_attributes) conditionnent le traçage.

vr_box dessine également un rectangle aux angles arrondis mais son traçage est conditionné par les attributs de ligne (Cf. vql_attributes).

int id_vdi, coord_xy[4] ;

« nom de la fonction » (id_vid, coord_xy) ;

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI
coord_xy[0] (ptsin[0]) : coord. X d'un coin de rectangle
coord_xy[1] (ptsin[1]) : coord. Y de ce même coin
coord_xy[2] (ptsin[2]) : coord. X du coin diagonalement opposé
coord_xy[3] (ptsin[3]) : coord. Y de ce même coin

v_circle fonction 11, sous-fonction 4

Dessine un disque. Les attributs de remplissage de zone affectent le traçage (cf vqf_attributes). Pour dessiner un cercle, utilisez v_arc.

int id_vdi, x, y, rayon ;

v_circle (id_vdi, x, y, rayon) ;

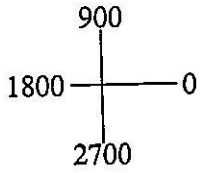
Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du centre du cercle
y (ptsin[1]) : coord. Y du centre du cercle
rayon (ptsin[4]) : taille du rayon en pixels

v_arc fonction 11, sous-fonction 2
v_pieslice fonction 11, sous-fonction 3

La fonction v_arc dessine un arc de cercle dont on spécifie le centre, le rayon, ainsi que les points de départ et d'arrivée. Les attributs de ligne conditionnent son traçage (cf. vqf_attributes).

La fonction v_pieslice dessine une portion de disque, avec des paramètres d'appel identiques. Ici cependant, ce sont les attributs de remplissage qui conditionnent le traçage (Cf vqf_attributes).

Pour ces deux fonctions, la même remarque générale s'impose :
Un angle se donne en dixième de degré en considérant la valeur 0 comme le point horizontal droit et en incrémentant dans le sens inverse des aiguilles d'une montre.




```
int id_vdi, x, y, rayon, debut, fin ;

v_arc (id_vdi, x, y, rayon, debut, fin) ;
v_pieslice (id_vdi, x, y, rayon, debut, fin) ;
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du centre de l'arc ou portion de cercle
y (ptsin[1]) : coord. Y du centre de l'arc ou portion de cercle
rayon (ptsin[6]) : rayon en pixels
debut (intin[0]) : départ de l'angle (compris entre 0 et 3600)
fin (intin[1]) : fin de l'angle (compris entre 0 et 3600)

v_ellipse fonction 11, sous-fonction 5
v_ellarc fonction 11, sous-fonction 6
v_ellpie fonction 11, sous-fonction 7

La fonction v_ellipse dessine une ellipse.
La fonction v_ellpie dessine une portion d'ellipse.
Ces deux fonctions sont affectées par les attributs de zone de remplissage (cf. vqf_attributes)

La fonction v_ellarc dessine un arc elliptique, elle est affectée par les attributs de traçage de ligne (cf. vql_attributes).

En ce qui concerne les paramètres d'angles pour v_ellarc et v_ellpie, voir la description dans la fonction précédente.

```
int id_vdi, x, y, xrayon, yrayon ;

int debut, fin ; /* pour v_ellarc et v_ellpie */

v_ellipse (id_vdi, x, y, xrayon, yrayon) ;
v_ellarc (id_vdi, x, y, xrayon, yrayon, debut, fin) ;
v_ellpie (id_vdi, x, y, xrayon, yrayon, debut, fin) ;
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du centre de l'arc (ou ellipse)
y (ptsin[1]) : coord. Y du centre de l'arc (ou ellipse)
xrayon (ptsin[2]) : rayon vertical en pixel
yrayon (ptsin[3]) : rayon horizontal en pixel

debut (intin[0]): départ de l'angle en dixième de degré (0 à 3600)
fin (intin[1]) : fin de l'angle en dixième de degré (0 à 3600)

Fonctions de remplissage de forme

vsf_interior fonction 23

Sélectionne le type de remplissage à utiliser pour tout remplissage ultérieur. Cinq types sont possibles :

0 = fond vide couleur du fond de l'écran (par défaut)
1 = fond plein couleur définie par vsf_color
2 = fond à motifs (cf vsf_style)
3 = fond hachuré (cf vsf_style)
4 = fond défini par l'utilisateur (cf vsf_udpat)

int id_vdi, type_remp, sel_remp ;

sel_remp = vsf_interior (id_vdi, type_remp) ;

Paramètres d'appel :

id_vdi (contrl[6]) ; identificateur VDI
type_remp (intin[0]) : numéro du type requis

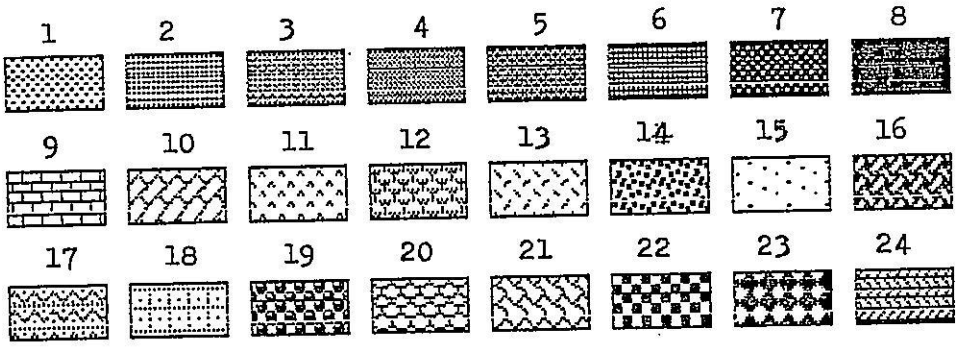
Paramètre de retour

sel_remp (intout[0]) : type effectivement sélectionné

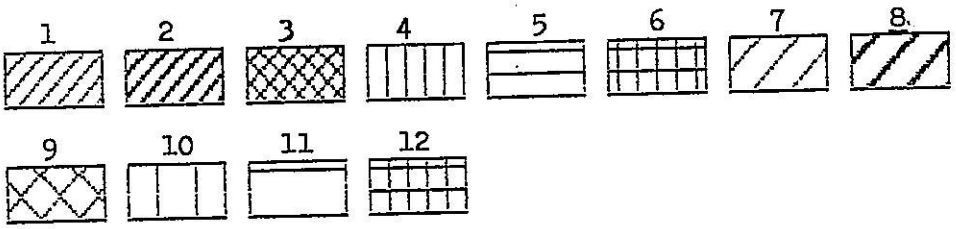
vsf_style fonction 24

Sélectionne une matrice qui définit le style de remplissage. Cette fonction n'est utilisable que pour les types de remplissage 2 et 3.

Avec le type de remplissage numéro 2 (motif)



Avec le type de remplissage numéro 3 (hachuré)



La couleur des matrices est définie par vsf_color.

```
int id_vdi, style_remp, sel_style;

sel_style = vsf_style(id_vdi, style_remp);
```

Paramètres d'appel

id_vdi (contrl[6]) : identificateur VDI
style_remp (intin[0]) : numéro de style requis

Paramètre de retour

sel_style (intout[0]) : style effectivement sélectionné

vsf_udpat fonction 112

Permet de créer un fond défini par l'utilisateur, appelable en sélectionnant le type de remplissage numéro 4 (cf vsf_interior).

Le masque de la matrice est spécifié dans 16 mots avec le bit 15 du premier mot représentant le bit du coin supérieur gauche, et le bit 0 du dernier mot représentant le bit du coin inférieur droit.

La basse et moyenne résolution utilise différents plans de masque pour définir la matrice (cf. chap. 3.3. instruction A004).

int id_vdi, plan, def_fond [16 * plan] ;

vsf_udpat (id_vdi, def_fond, plan) ;

Paramètres d'appel

id_vdi (contrl [6]) : identificateur VDI
plan (contrl[3]/16) : nombre de plans
def_fond[0] (intin[0]) : premier mot de définition matrice
...
def_fond[15] (intin[15]) : dernier mot du premier plan

Si plusieurs plans, alors...
def_fond[16] (intin[16]) : premier mot du deuxième plan
...
def_fond[16*plan – 1] (intin[16*plan – 1]) : dernier mot du dernier plan

vsf_perimeter fonction 104

Indique au VDI si, lors d'un remplissage, une zone doit ou non être entourée d'une ligne de contour. Ligne invisible, puisqu'elle est de la couleur du remplissage !

```
int id_vdi, peri_drap, sel_peri;

sel_peri = vsf_perimeter(id_vdi, peri_drap);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI
peri_drap (intin[0]) : drapeau de périmètre de zone
0 = périmètre invisible, sinon = visible

Paramètre de retour :

sel_peri (intout[0]) : drapeau effectivement sélectionné

v_fillarea fonction 9

Cette fonction colorie un polygone quelconque délimité par des points (64 au maximum), le dernier de ceux-ci rejoignant le premier pour clore la surface à remplir (voir 3.3. instruction A00F).

Utilise les attributs de remplissage (cf vqf_attributes).

```
int id_vdi, nb_point, coord_xy [2 * nb_point];

v_fillarea(id_vdi, nb_point, coord_xy);
```

id_vdi (contrl[6]) : identificateur VDI
nb_point (contrl[1]) : nombre de points du polygone
coord_xy[0] (ptsin[0]) : coord. X du premier point
coord_xy[1] (ptsin[1]) : coord. Y du premier point
... ...

et en considérant un polygone à N points,

coord_xy[2N−2] (ptsin[2N−2]) : coord. X du dernier point
coord_xy[2N−1] (ptsin[2N−1]) : coord. Y du dernier point

v_contourfill fonction 103

Remplit, à partir d'un point précisé, une surface délimitée soit par le bord de l'écran, soit par les pixels de la couleur d'index spécifié. Si ce dernier a une valeur négative, tout pixel d'une couleur différente de la zone à remplir constitue une « barrière » (cf. chap. 3.3. instruction A00F).

```
int id_vdi, x, y, index;

v_contourfill(id_vdi, x, y, index);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du point de départ de remplissage
y (ptsin[1]) : coord. Y du point de départ du remplissage
index (intin[0]) : numéro de la couleur de clôture

vqf_attributes fonction 37

Cette fonction retourne les attributs qui affectent tout remplissage de zone.

```
int id_vdi, attrib [4];

vqf_attrtibutes( id_vdi, attrib);
```

Paramètre d'appel

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :

attrib[0] (intout[0]) : type de remplissage (cf vsf_interior)
attrib[1] (intout[1]) : couleur de remplissage (cf vsf_color)
attrib[2] (intout[2]) : style de remplissage (cf vsf_style)
attrib[3] (intout[3]) : mode d'écriture (cf vswr_mode)

Fonctions de marqueurs

Les marqueurs sont des formes pré-définies, de taille restreinte, susceptibles d'être utilisés pour agrémenter le graphisme.

vsm_type fonction 18

Sélectionne le type de marqueurs pour tout affichage ultérieur. Six types sont possibles :

1 = . un point.
2 = + le signe plus
3 = * l'astérisque
4 = □ le carré
5 = X la croix
6 = ◊ le losange

```
int id_vdi, type_mar, sel_mar;

sel_mar = vsm_type(id_vdi, type_mar);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
type_mar (intin[0]) : type de marqueur requis

Paramètre de retour :

sel_mar (intout[0]) : type effectivement sélectionné

vsm_height fonction 19

Sélectionne la hauteur du marqueur (sauf le point, qui ne peut être modifié), parmi 8 tailles comprises entre 11 et 88 pixels. Le VDI retourne la plus proche (et inférieure) taille valide. La largeur est modifiée en proportion.

```
int id_vdi, haut_mar, sel_hmar;

sel_hmar = vsm_height(id_vdi, haut_mar);
```

Paramètres d'appel

id_vdi (contrl[6]) : identificateur VDI
haut_mar (ptsin[1]) : hauteur requise du marqueur en pixels

Paramètre de retour :

sel_hmar (ptsout[1]) : hauteur effectivement sélectionnée

v_pmarker fonction 7

Dessine un ou des marqueurs aux coordonnées spécifiées, selon les attributs définis (cf. vqm_attributes).

```
int id_vdi, nb, coord_xy[2 * nb_mar];

v_pmarker(id_vdi, nb, coord_xy);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI
nb (contrl[1]) : nombre de marqueurs à tracer
coord_xy[0] (ptsin[0]) : coord. X du premier marqueur
coord_xy[1] (ptsin[1]) : coord. Y du premier marqueur
...
coord_xy[2nb – 2](ptsin[2nb – 2]) : coord. X du dernier marqueur
coord_xy[2nb – 1] (ptsin[2nb – 1]) : coord. Y du dernier marqueur

vqm_attributes fonction 36

Retourne les attributs qui affectent le traçage des marqueurs.

```
int id_vdi, attrib[4];

vqm_attributes(id_vdi, attrib);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :

attrib[0] (intout[0]) : type du marqueur (cf vsm_type)
attrib[1] (intout[1]) : couleur du marqueur (cf vsm_color)
attrib[2] (intout[2]) : mode d'écriture (cf vswr_mode)
attrib[3] (ptsout[1]) : hauteur du marqueur (cf vsm_height)

Fonctions de couleurs

vs_color fonction 14

Cette fonction permet d'associer un numéro d'index avec un pourcentage de Rouge, Vert, Bleu, créant ainsi une palette utilisateur (cf. chap. 1.1.2. relatif au système graphique).

Les seize index permis prennent les valeurs de 0 à 15, l'index 0 étant considéré par le système comme la couleur de fond.

L'intensité des couleurs est définie par une valeur comprise entre 0 et 1 000, mais la valeur donnée sera arrondie par la fonction pour l'adapter à une codage sur 3 bits. Tout écart de l'éventail de valeurs autorisées entraîne un réajustement aux valeurs extrêmes (0 si < 0, 1 000 si > 1 000).

```
int id_vdi, index, rgb_in[3];

vs_color(id_vdi, index, rgb_in);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
index (intin[0]) : numéro d'index
rgb_in[0] (intin[1]) : pourcentage de rouge (entre 0 et 1 000)
rgb_in[1] (intin[2]) : pourcentage de vert
rgb_in[2] (intin[3]) : pourcentage de bleu

vsl_color fonction 17. (ligne)
vsf_color fonction 25. (remplissage)
vsm_color fonction 20. (marqueurs)
vst_color fonction 22. (texte)

Ces quatre fonctions sélectionnent l'index de couleur qui sera utilisé par le type de fonction précisé entre parenthèse.

La couleur par défaut est celle d'indice 1.

```
int id_vdi, ind_coul, sel_coul;

sel_coul = "nom de la fonction" (id_vdi, ind_coul);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
ind_coul (intin[0]) : index de la couleur (cf. vs_color)

Paramètre de retour :

sel_coul (intout[0]) : index effectivement sélectionné

v_get_pixel fonction 105

Retourne la valeur et index de la couleur d'un pixel spécifié. Voir chap. 1.1.2. pour la correspondance entre couleur et index.

```
int id_vdi, x, y, *val_col, *index;

v_get_pixel(id_vdi, x, y, val_col, index);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du pixel
y (ptsin[1]) : coord. Y du pixel

Paramètres de retour :

val_col (intout[0]) : valeur de la couleur
index (intout[1]) : index de la couleur

vq_color fonction 26

Cette fonction retourne soit la valeur RVB requise (lors d'un vs_color) d'un index de couleur, soit la valeur arrondie réellement placée par le système pour ce même index. Le choix de la valeur à recevoir est fonction de l'état du drapeau.

```
int id_vdi, index_coul, drapeau, rgb [3];

vq_color(id_vdi, index_coul, drapeau, rgb);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
index_coul (intin[0]) : index de la couleur
drapeau (intin[1]) : si mis à 0 retourne la valeur requise
si mis à 1 retourne la valeur réelle arrondie

Paramètres de retour

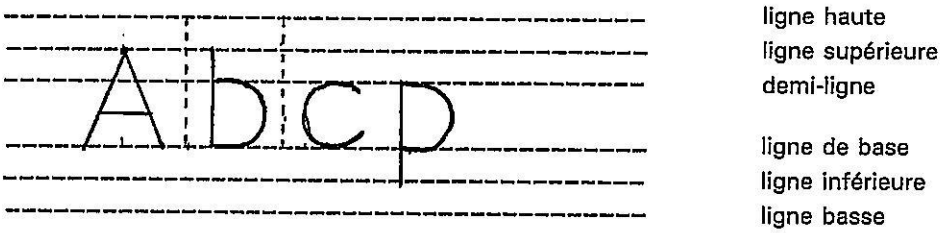
coul (intout[0]) : index de couleur
rgb[0] (intout[1]) : pourcentage de rouge dans la couleur
rgb[1] (intout[2]) : pourcentage de vert dans la couleur
rgb[2] (intout[3]) : pourcentage de bleu dans la couleur

Affichage de texte en mode graphique

Le mode graphique est le mode par défaut lors de l'ouverture d'une station de travail. Tout affichage dans ce mode réfère à la position du curseur graphique.

vst_alignment fonction 39

Cette fonction définit le type d'alignement (horizontal et vertical) que le VDI utilisera lorsque lui sera demandé un affichage de texte à la position du curseur graphique.



Par défaut, c'est l'alignement horizontal à gauche et l'alignement vertical sur la ligne de base qui sont sélectionnés.

```
int id_vdi, hor_in, vert_in, hor_out, vert_out ;

vst_alignment (id_vdi, hor_in, vert_in, &hor_out, &vert_out) ;
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
hor_in (intin[0]) : alignement horizontal requis (3 valeurs possibles)
0 = affichage à la gauche du curseur
1 = affichage centré sur le curseur
2 = affichage à la droite du curseur

vert_in (intin[1]) : alignement vertical requis (5 valeurs possibles)
0 = la coord. Y du curseur correspond à la « ligne de base »
1 = la coord. Y du curseur correspond à la « demi-ligne »
2 = la coord. Y du curseur correspond à la « ligne supérieure »
3 = la coord. Y du curseur correspond à la « ligne basse »
4 = la coord. Y du curseur correspond à la « ligne inférieure »
5 = la coord. Y du curseur correspond à la « ligne haute »

Paramètres de retour :

hor_out (intout[0]) : alignement horizontal sélectionné
vert_out (intout[1]) : alignement vertical sélectionné

vst_height fonction 12

Cette fonction spécifie la hauteur des caractères, en fait la distance entre la ligne de base et la ligne haute (voir schéma ci-dessus).

Le VDI ajustera la hauteur demandée au plus proche de ses hauteurs et déterminera la largeur du caractère en proportion.

La hauteur minimale est de 4 pixels.

```
int id_vdi, haut_car, car_larg, car_haut, cel_larg, cel_haut;

vst_height (id_vdi, haut_car, &car_larg, &car_haut,
                        &cel_larg, &cel_haut);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
haut_car (ptsin[1]) : hauteur de caractère requise

Paramètres de retour

car_larg (ptsout[0]) : largeur de caractère sélectionnée
car_haut (ptsout[1]) : hauteur de caractère sélectionnée
cel_larg (ptsout[2]) : largeur de cellule sélectionnée
cel_haut (ptsout[3]) : hauteur de cellule sélectionnée

vst_rotation fonction 13

Spécifie l'angle de rotation de la ligne de base du caractère. Les 4 valeurs permises sont, en dixième de degré : 0, 900, 1 800, 2 700.

```
int sel_base, id_vdi, angle;

sel_base = vst_rotation (id_vdi, angle);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI
angle (intin[0]) : l'angle de rotation en dixièmes de degré

Paramètre de retour :

sel_base (intout[0]) : l'angle effectivement sélectionné

vst_effects fonction 106

Cette fonction permet de spécifier un ou plusieurs effets spéciaux sur les caractères. Ces effets sont sélectionnés si le bit leur correspondant est mis à 1.

bit 0 = caractère épaissi
bit 1 = caractère à intensité réduite
bit 2 = caractère oblique
bit 3 = caractère souligné
bit 4 = caractère esquissé

Exemple : la valeur $B (1011 en binaire) dans « effet » spécifiera un caractère épaissi, à intensité réduite, et souligné.

```
int id_vdi, sel_effet, effet;

sel_effet = vst_effects (id_vdi, effet);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
effet (intin[0]) : valeur de l'effet

Paramètre de retour :

sel_effet (intout[0]) : effets sélectionnés

v_gtext fonction 8

Cette fonction affiche un ou plusieurs caractères sur l'écran graphique, à la position spécifiée par ses coordonnées. L'affichage est conditionné par les différents attributs de texte (cf. vqt_attributes).

Les codes ASCII de la chaîne à afficher sont stockés dans les octets faibles du tableau d'entrée « intin ».

```
int id_vdi, x, y;
char chaine [n + 1];      /* n = nombre de caractères de la chaine */
v_gtext (id_vdi, x, y, chaine);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du point d'affichage
y (ptsin[1]) : coord. Y du point d'affichage
chaîne[0] (intin[0]) : premier caractère de la chaîne
... ...
chaîne[n] (intin[n]) : valeur 0 indiquant la fin de chaîne

v_justified fonction 11, sous-fonction 10

Affiche une chaîne de caractères à l'écran en tentant, selon les facultés qui lui sont laissées, de la justifier à gauche et à droite. Notez que cette fonction s'utilise avec un code de sous-fonction en contrl [5].

Les attributs de texte conditionne l'affichage (cf. vqt_attributes).

```
int id_vdi, x, y, long, aut_mot, aut_carac;
char chaine [n + 1];   /* n = longueur de chaine */

v_justified (id_vdi, x, y, chaine, long, aut_mot, aut_carac);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
x (ptsin[0]) : coord. X du point d'affichage
y (ptsin[1]) : coord. Y du point d'affichage
long (ptsin[2]) : longueur, en pixels, laissée pour l'affichage
aut_mot (intin[0]) : si non-zéro, autorise le VDI à modifier l'espacement entre les mots. Si 0, alors interdiction
aut_carac (intin[1]) : si non-zéro, autorise le VDI à modifier l'espacement entre les caractères. Si 0, alors interdiction

chaîne[0] (intin[2]) : premier caractère de la chaîne

... ...

chaîne[n] (intin[n+2]) : valeur 0 indiquant la fin de chaîne

vqt_extent fonction 116

Cette fonction retourne les coordonnées d'un rectangle représentant l'encadrement de la chaîne spécifiée. Tracer des lignes entre ces points permet donc d'entourer la chaîne de caractère.

Les quatre points retournés ont des coordonnées relatives à la chaîne et ne tiennent donc pas compte de la position de celle-ci sur l'écran.

Le point 1 indique le coin inférieur gauche du rectangle,
Le point 2 indique le coin inférieur droit,
Le point 3 indique le coin supérieur droit,
Le point 4 indique le coin supérieur gauche.

```
int id_vdi, rect [8];
char chaine [n + 1]; /* n = nombre de caractères de la chaine */

vqt_extent (id_vdi, chaine, rect);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
chaîne[0] (intin[0]) : premier caractère de la chaîne

... ...

chaîne[n] (intin[n]) : valeur 0 indiquant la fin de la chaîne

Paramètres de retour :

rect[0] (ptsout[0]) : coord. relative X du point 1
rect[1] (ptsout[1]) : coord. relative Y du point 1

... ...

rect[6] (ptsout[6]) : coord. relative X du point 4
rect[7] (ptsout[7]) : coord. relative Y du point 4

vqt_width fonction 117

Retourne la largeur, en pixels, de la cellule et des demi-espacements gauche et droit, (cf. schéma dans vst_alignment) d'un caractère spécifié.

La fonction ne tient pas compte de l'influence éventuelle des effets spéciaux sur les largeurs retournées.

```
int id_vdi, cel_larg, gau_esp, droit_esp, erreur;
char carac;

erreur = vqt_width (id_vdi, carac, &cel_larg, &gau_esp,
                                &droit_esp);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
carac (intin[0]) : code ASCII du caractère

Paramètres de retour :

erreur (intout[0]) : −1 si caractère non valide sinon, code ASCII du caractère requis
cel_larg (ptsout[0]) : largeur de la cellule
gau_esp (ptsout[2]) : demi-espacement gauche
droit_esp (ptsout[4]) : demi-espacement droit

vqt_attributes fonction 38

Cette fonction retourne les attributs courants qui affectent l'affichage du texte graphique.

```
int id_vdi, attrib [10];

vqt_attributes (id_vdi, attrib);
```

Paramètre d'appel

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour

attrib[0] (intout[0]) : police de caractères (cf vst_load_fonts)
attrib[1] (intout[1]) : couleur du texte (cf vst_color)
attrib[2] (intout[2]) : angle d'inclinaison du texte (cf vst_rotation)
attrib[3] (intout[3]) : alignement horizontal (cf vst_alignment
attrib[4] (intout[4]) : alignement vertical (cf. vst_alignment)
attrib[5] (intout[5]) : mode d'écriture (cf. vswr_mode)
attrib[6] (ptsout[0]) : largeur des caractères
attrib[7] (ptsout[1]) : hauteur des caractères
attrib[8] (ptsout[2]) : largeur de la cellule de caractère
attrib[9] (ptsout[3]) : hauteur de la cellule de caractère

Fonctions texte en mode alphanumérique

Le VDI dispose d'un véritable éditeur (celui dont se sert d'ailleurs l'accessoire de bureau « VT.52 »), riche d'un grand nombre de commandes internes dont voici les principales :

1) Les caractères de commandes compris entre $07 et $0D (correspondant à l'appui simultané de la touche « CONTROL » et, respectivement, des touches « G » à « M ») sont valides.

Ex : CONTROL-J (c'est-à-dire $0A) provoque le passage du curseur à la ligne suivante.

2) Les différents caractères précédés de l'appui de la touche «ESCAPE » ont les fonctions suivantes :

A : curseur vers le haut.
B : curseur vers le bas.
C : curseur vers la droite.
D : curseur vers la gauche.
E : CLEAR-HOME
F : HOME
d : efface curseur → haut écran.
J : efface curseur → fin écran.
K : efface curseur → fin ligne.
o : efface début ligne → curseur.
M : efface ligne.
L : insert de ligne.
e : activation curseur
f : désactivation curseur.
p : vidéo inverse.
q : vidéo normale.

v : autorise le retour de ligne automatique.
w : interdit le retour de ligne automatique.
j : « marque » la position du curseur.
k : positionne le curseur à la « marque » de ESC-j.
b (+ 1 paramètre) : quartet faible du paramètre = couleur du caractère.
c : (+ 1 paramètre) : quartet faible du paramètre = couleur de cellule.
y : (+ 2 paramètres) : envoie le curseur à la position absolue spécifiée par la valeur des paramètres moins $20.

Ex : ESC-Y # 0 (respectivement $23, $30) positionne le curseur à la troisième rangée, dixième colonne.

Les fonctions suivantes s'adressent directement à l'éditeur VDI. Elles utilisent toutes le numéro de fonction 5 (appelée ESCAPE par le GEM), et s'identifient donc par le numéro de sous-fonction.

v_enter_cur fonction 5, sous-fonction 3

Cette fonction génère l'entrée dans l'éditeur, provoque l'activation du curseur alpha dans le coin supérieur de l'écran, désactive le mode graphique.

```
int id_vdi;

v_enter_cur (id_vdi);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI

v_curup fonction 5, sous-fonction 4
v_curdown fonction 5, sous-fonction 5
v_curright fonction 5, sous-fonction 6
v_curleft fonction 5, sous-fonction 7
v_curhome fonction 5, sous-fonction 8

Ces 5 fonctions provoquent un déplacement du curseur alphanumérique,
à la rangée supérieure pour v_curup
à la rangée inférieure pour v_curdown
une colonne à droite pour v_curright
une colonne à gauche pour v_curleft
à sa position initiale pour v_curhome (coin haut à gauche)

```
int id_vdi;

"nom de la fonction" (id_vdi);
```

Paramètre d'appel : id_vdi (contrl[6]) : identificateur VDI

v_rvon fonction 5, sous-fonction 13
v_rvoff fonction 5, sous-fonction 14

v_rvon affiche tout texte ultérieur en vidéo inverse, v_rvoff annule l'effet précédent.

int id_vdi ;

« Nom de la fonction » (id_vdi) ;

Paramètre d'appel : id_vdi (contrl[6]) : identificateur VDI

v_eeos fonction 5, sous fonction 9
v_eeol fonction 5, sous-fonction 10

La fonction v_eeos efface depuis curseur alpha jusqu'à fin d'écran.
La fonction v_eeol efface depuis curseur alpha jusqu'à fin de ligne.

```
int id_vdi;

"nom de la fonction" (id_vdi);
```

Paramètre d'appel : id_vdi (contrl[6]) : identificateur VDI

vs_curaddress fonction 5, sous-fonction 11

Déplace le curseur alpha à la position spécifiée.

```
int id_vdi, rangée, colonne;

vs_curaddress (id_vdi, rangee, colonne);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
rangee (intin[0]) : numéro de la rangée (1 minimum)
colonne (intin[1]) : numéro de la colonne (1 minimum)

v_curtex fonction 5, sous-fonction 12

Affiche une chaîne de caractères à la position du curseur alpha.

```
int id_vdi ;
char chaîne [n + 1] ; /* n = longueur de chaîne */

v_curtext (id_vdi, chaîne) ;
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
chaîne[0] (intin[0]) : premier caractère de la chaîne
... ...
chaîne[n] (intin[n]) : valeur 0 terminant la chaîne

vq_curaddress fonction 5, sous-fonction 15

Retourne la position du curseur alpha.

```
int id_vdi, rangée, colonne;

vq_curaddress (id_vdi, &rangée, &colonne);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :

rangee (intout[0]) : numéro de rangée
colonne (intout[1]) : numéro de colonne

vq_chcells fonction 5, sous-fonction 1

Retourne le nombre de positions, en rangées et en colonnes, auxquelles le curseur alpha peut se placer. La réponse dépend de la taille de la cellule de caractère utilisée.

```
int id_vdi, rangée, colonne;

vq_chcells (id_vdi, &rangée, &colonne);
```

Paramètre d'appel :
id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :
rangee (intout[0]) : nombre de rangées adressables
colonne (intout[1]) : nombre de colonnes adressables

ESCAPE 101

Utilisable uniquement en assembleur : décale le curseur ALPHA en multipliant le nombre de caractères par ligne, par l'argument mis en intin[0].

ESCAPE 102

Initialise une police de caractères utilisable par l'éditeur VDI.

Cette fonction n'a pas de nom reconnu par l'éditeur de liens et ne peut donc être appelée en C. Dans le chapitre 3.1.3. est listée la routine assembleur permettant d'accéder à cette fonction.

v_exit_cur fonction 5, sous-fonction 2

Sortie de l'éditeur VDI et réactivation du mode graphique

```
int id_vdi;

v_exit_cur (id_vdi);
```

Paramètre d'appel : id_vdi (contrl[6]) : identificateur VDI

Fonctions formes souris

vsc_form fonction 111

Permet de redéfinir la forme de la souris en spécifiant bits de données et de masques, et point-souris (c. chap. 3.3. instr. A00B).

Pour données et masque, le bit 15 du premier mot correspond au coin supérieur gauche de la forme. Le point-souris indique le point de la forme devant être considéré comme coordonnées X et Y de la souris (se donne relativement au coin supérieur gauche de la forme).

```
int id_vdi, sour_for [37];

vsc_form (id_vdi, sour_for);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
sour_for[0] (intin[0]) : coord. X du point-souris.
sour_for[1] (intin[1]) : coord. Y du point-souris.
sour_for[2] (intin[2]) : réservé, doit être à 1.
sour_for[3] (intin[3]) : index couleur du masque (en théorie à 0).
sour_for[4] (intin[4]) : index couleur données (en théorie à 1).
sour_for[5] à [20] (intin[5] à [20]) : mots du masque.
sour_for[21] à [36] (intin[21] à [36]) : mots des données.

v_hide_c fonction 123

Cache la souris et décrémente un compteur (cf. chap. 3.3. instructions A009 et A00A).

```
int id_vdi;

v_hide_c (id_vdi);
```

Paramètre d'appel : id_vdi (contrl[6]) : identif. VDI.

v_show_c fonction 122

Fait apparaître la souris si compteur = 0, ou s'il est spécifié de ne pas tenir compte de ce compteur (cf. chap. 3.3. instructions A009, A00A).

```
int id_vdi, dp_cpt;

v_show_c (id_vdi, dp_cpt);
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI.
dp_cpt (intin[0]) : si 0, ne se soucie pas du compteur.

v_dspcur fonction 5, sous-fonction 18

Visualise la souris aux coordonnées spécifiées.

```
int id_vdi, x, y;

v_dspcur (id_vdi, x, y);
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI.
x (ptsin[0]) : coord. X du point d'affichage.
y (ptsin[1]) : coord. Y du point d'affichage.

v_rmcur fonction 5, sous-fonction 19

Supprime la dernière souris affichée.

```
int id_vdi;

v_rmcur (id_vdi);
```

Paramètre d'appel : id _vdi (contrl [6]) : identificateur VDI.

Fonctions d'entrées souris et clavier

Le VDI permet d'interroger les périphériques souris et clavier en utilisant deux « modes d'interrogations » différents :

— le mode « attente » (request) qui ne revient de la fonction que lorsqu'une entrée a effectivement eu lieu.
— le mode « test » (sample) qui interroge le périphérique et retourne, sans attendre, l'état de celui-ci.

Dans quatre « genres d'entrée » que le GEM peut implémenter, seuls deux sont disponibles dans la version 520 ST :

— « locator » qui, s'adressant à la souris, retourne une localisation.
— « string » qui retourne une chaîne du clavier.

Le genre « valuator » (qui gère uniquement les touches de direction-clavier) n'est pas implémenté, « choice » qui permet de spécifier une alternative retourne toujours la valeur 1 et n'est donc pas utilisable.

vsin_mode fonction 33

Permet de spécifier le mode d'interrogation requis pour un genre d'entrée donné.

```
int id_vdi, genre, mode;

vsin_mode (id_vdi, genre, mode);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
genre (intin[0]) : 1 (« locator » : souris), 4 (« string » : clavier)
mode (intin[1]) : si 1 = « attente », si 2 = « test ».

vrq_locator (ou vsm_locator) fonction 28

Interroge la souris selon le mode défini par vsin_mode. Les touches de direction du clavier (qui permettent également le déplacement de la souris) sont testées. En mode « attente », la souris est affichée.

```
int id_vdi, etat, x, y, xout, yout, rep;

etat = vrq(sm)_locator (id_vdi, x, y, &xout, &yout, &rep);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI.
x (ptsin[0]) : coord. X initiale de la souris
y (ptsin[1]) : coord. Y initiale de la souris

Paramètres de retour :

état (contr[2] | contrl[4]<<1) : à 0 (mode « test ») si aucun déplacement, à 1 si déplacement souris, à 2 si bouton pressé, à 3 si (1 + 2).
xout (ptsout[0]) : nouvelle coord. X de la souris.
yout (ptsout[1]) : nouvelle coord. Y de la souris.
rep (intout[0]) : si bouton, valeur bouton (cf. vq_mouse) + $20
Si touche clavier, ASCII dans l'octet faible, code de la touche (cf TRAP 1, fonction 1) dans l'octet fort.

vrq_string (vsm_string) fonction 31

Retourne une chaîne du clavier selon le mode défini par vsin_mode.

En mode « attente », les caractéristiques entrés au clavier sont stockés tant que le tampon spécifié n'est pas plein ou que la touche RETURN n'est pas pressée.

En mode « test », c'est le tampon-clavier du système qui est lu et son contenu renvoyé, avec les mêmes arrêtoirs que pour le mode « attente ».

```
int etat, id_vdi, max_lg, dp_ec, xy_ec [2];
char chaine [N];

etat = vrq(sm)_string (id_vdi, max_lg, dp_ec, xy_ec, chaine);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
max_lg (intin[0]) : longueur de chaîne requise (si valeur < 0, la longueur de chaîne maximum = longueur du tableau de sortie intout[]).
xy_ec[0] (ptsin[0]) : coord. X de l'affichage de l'écho.
xy_ec[1] (ptsin[1]) : coord. Y de l'affichage de l'écho.

Paramètres de retour :

chaîne[0] à chaîne[N – 1] (intout[0] à intout [N – 1] :
code ASCII de la chaîne reçue (terminée par la valeur nulle).

vq_mouse fonction 124

Retourne la position de la souris et l'état courant de ses boutons.

```
int id_vdi, etat_bt, xsour, ysour;

vq_mouse (id_vdi, etat_bt, xsour, ysour);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI.

Paramètres de retour :

état_bt (intout[0]) : le bit 0 correspond au bouton le plus à gauche, le bit 1 au second bouton en partant de la gauche, etc...
Si le bit est mis à 1, alors bouton correspondant enfonçé.
xsour (ptsout[0]) : coord. X de la souris.
ysour (ptsout[1]) : coord. Y de la souris.

vq_key _s fonction 128

Retourne l'état courant des touches SHIFT, CONTROL et ALTERNATE.

```
int id_vdi, etat;

vq_key_s (id_vdi, &etat);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI.

Paramètre de retour :

état (intout[0]) : si le bit est à 1, la touche est pressée.
bit 0 = SHIFT (droite), bit 1 = SHIFT (gauche), bit 2 = CONT,
bit 3 = ALT

Fonctions de copie de zones

Ces fonctions opèrent des copies de zones rectangulaires, de mémoire à mémoire et octet par octet, depuis une source vers une destination, en utilisant une opération logique spécifique.

Ces copies sont fréquemment utilisées soit dans le sens « mémoire d'écran vers mémoire tampon » afin de sauvegarder une image, soit dans le sens contraire afin de restituer l'image précédemment sauvegardée.

Chaque argument de la copie, qu'il soit source ou destination, est défini par une structure MFDB, spécifiant son adresse, sa taille, l'opération logique à effectuer, etc... (cf. instr. A00E, chap. 3.3.).

vro_cpyfm fonction 109

Copie une zone source rectangulaire vers une zone destination (les deux blocs de bits source et destination étant définis par une structure MFDB) en fournissant comme arguments, l'opération logique spécifiée, les coordonnées des rectangles source et destination, ainsi que les adresses des deux structures MFDB (cf. chap. 3.3. instr. A007 et A00E).

Si zone source et zone destination ont des coordonnées différentes, la copie est indicée sur la source.

```
int id_vdi, op_log, coord_xy [8];
FDB *source, *dest;

vro_cpyfm (id_vdi, op_log, coord_xy, source, dest);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
op_log (intin[0]) : opération logique entre source et destination
coord_xy[0] (ptsin[0]) : coord. X d'un coin du rectangle source
coord._xy[1] (ptsin[1]) : coord. Y de ce même coin
coord_xy[2] (ptsin[2]) : coord. X du coin du rectangle source
diagonalement opposé par rapport à coord_xy[0]
coord_xy[3] (ptsin[3]) : coord;. Y de ce même coin
coord_xy[4] (ptsin[4]) : coord. X d'un coin du rectangle destination
coord_xy[5] (ptsin[5]) : coord. Y de ce même coin
coord_xy[6] (ptsin[6]) : coord. X du coin du rectangle destination
diagonalement opposé par rapport à coord_xy[4]
coord_xy[7] (ptsin[7]) : coord. Y de ce même coin
source (contrl[7] et [8]) : adresse pointant sur la structure FDB source
dest (contrl[9] et [10]) : adresse pointant sur la structure FDB destination

vrt_cpyfm fonction 121

Cette fonction, variante de la précédente, copie une zone source monochrome, en transformant, dans la destination, les deux couleurs de base par les couleurs dont l'index est spécifié.

Seules les quatre premières opérations logiques sont autorisées.

```
int id_vdi, op_log, coord_xy [8], index [2];
FDB *source. *dest;

vrt_cpyfm (id_vdi, op_log, coord_xy, source, dest, index);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
op_log (intin[0]) : opération logique entre source et destination
index[0](intin[1]) : index de couleur remplaçant les pixels 1 de la source
index[1](intin[2]) : index de couleur remplaçant les pixels 0 de la source
coord_xy[0] (ptsin[0]) : coord. X d'un coin du rectangle source
coord_xy[1] (ptsin[1]) : coord. Y de ce même coin
coord_xy[2] (ptsin[2]) : coord. X du coin du rectangle source
diagonalement opposé par rapport à coord_xy[0]

coord_xy[3] (ptsin[3]) : coord. Y de ce même coin
coord_xy[4] (ptsin[4]) : coord. X d'un coin du rectangle destination
coord_xy[5] (ptsin[5]) : coord. Y de ce même coin
coord_xy[6] (ptsin[6]) : coord. X du coin du rectangle destination diagonalement opposé par rapport à coord_xy[4]
coord_xy[7] (ptsin[7]) : coord. Y de ce même coin
source (contrl[7] et [8]) : adresse de la structure MFDB source
dest (contrl[9] et [10]) : adresse de la structure MFDB destination

vr_trnfm fonction 110

Recopie une structure MFDB source dans une MFDB destination en basculant le drapeau fd-stand (cf chap. 3.3. instr. A0007) transformant ainsi un bloc dépendant du périphérique en bloc standard, ou réciproquement.

```
int id_vdi;
FDB *source, *dest;

vr_trnfm (id_vdi, source, dest);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI
source (contrl[7] et [8]) : adresse de la structure MFDB source
dest (contrl[9] et [10]) : adresse de la structure MFDB destination

Fonctions de police de caractères

vst_load_fonts fonction 119

Cette fonction initialise les paramètres d'une ou plusieurs polices utilisateur. L'appel en C s'effectuant de façon incomplète, il est nécessaire d'ajouter des arguments par une routine assembleur (cf. chap. 3.1.3.). Retourne un identificateur de police(s).

```
int id_vdi, zero, id_pol;

id_pol = vst_load_fonts (id_vdi, zero);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI.
zero (intin[0]) : réservé, doit être à 0.

Paramètre de retour :

id_pol (intout[0]) : identificateur de la nouvelle police.

vst_font fonction 21

Sélectionne la police à employer pour toute écriture graphique ultérieure.

```
int id_vdi, id_pol, sel_pol;

sel_pol = vst_font (id_vdi, id_pol);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI.
id_pol (intin[0]) : identificateur de police (1 = police-système).

Paramètre de retour :

sel_pol (intout[0]) : la police effectivement sélectionnée.

vqt_name fonction 130

Retourne une chaîne de 32 caractères précisant le nom et le style de la police spécifiée.

```
int id_vdi, nu_ord, id_pol;
char nom [32];

id_pol = vqt_name (id_vdi, nu_ord, nom);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI.
nu_ord (intin[0]) : numéro d'ordre (1 pour police système).

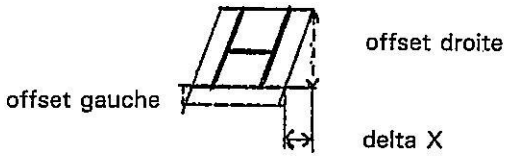
Paramètres de retour :

id_pol (intout[0]) : identificateur de police.
name[0] à [15] (intout[1] à [16]) : nom (en ASCII).
name[16] à [31] (intout[17] à [32]) : le style (en ASCII).

vqt_fontinfo fonction 131

Retourne des informations sur la police de caractère courante, en tenant compte de la taille (vst_height) et des effets spéciaux (vst_effects) sélectionnés.

Voir également le schéma
dans vst_alignment.




```
int id_vdi, mink, maxk, max_cel, rens [5], effet [3];

vqt_fontinfo (id_vdi, &mink, &maxk, rens, &max_cel, effet);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI

Paramètres de retour :

mink (intout[0]) : code ASCII du premier caractère dans la police.
maxk (intout[1]) : code ASCII du dernier caractère dans la police.
max_cel (ptsout[0]) : largeur max. de cellule (effets non-inclus).
rens[0] (ptsout[1]) : écart entre ligne basse et ligne de base.
rens[1] (ptsout[3]) : écart entre ligne inférieure et ligne de base.
rens[2] (ptsout[5]) : écart entre demi-ligne et ligne de base.
rens[3] (ptsout[7]) : écart entre ligne supérieure et ligne de base.
rens[4] (ptsout[9]) : écart entre ligne haute et ligne de base.
effet[0] (ptsout[2]) : delta X (voir schéma ci-dessus).
effet[1] (ptsout[4]) : offset gauche.
effet[2] (ptsout[6]) : offset droit.

vst_unload_fonts fonction 120

Désactive la (ou les) police(s) initialisées par vst_load_fonts.

```
int id_vdi, id_pol;

vst_unload_fonts (id_vdi, id_pol);
```

Paramètre d'appel :

id_vdi (contrl[6]) : identificateur VDI.
id_pol (intin[0]) : identificateur de la police à désactiver.

Fonctions d'imprimante

vst_point fonction 107

Spécifie, en points imprimables, la hauteur de cellule du caractère graphique. Un point est équivalent à 1/72 de pouce (soit 0.353 millimètres).

```
int id_vdi, sel, point, k_lg, k_ht, cel_lg, cel_ht;

sel = vst_point (id_vdi, point, &k_lg, &k_ht, &cel_lg, &cel_ht);
```

Paramètres d'appel :
id_vdi (contrl[6]) : identificateur VDI.
point (intin[0]) : nombre de points définissant hauteur cellule.

Paramètres de retour :
sel (intout[0]) : hauteur sélectionnée, en points.
k_lg (ptsout[0]) : largeur du caractère en pixels.
k_ht (ptsout[1]) : hauteur du caractère en pixels.
cel_lg (ptsout[2]) : largeur de cellule en pixels.
cel_ht (ptsout[3]) : hauteur de cellule en pixels

v_hardcopy fonction 5, sous-fonction 17

Recopie sur imprimante, point par point, l'intégralité de l'écran.

```
int id_vdi;

v_hardcopy (id_vdi);
```

Paramètre d'appel : id_vdi (contrl[6]) : identificateur VDI.

Fonctions vectorisées

Ces fonctions permettent de modifier les adresses de branchement de certaines routines d'interruptions et vous autorisent donc à en prendre le contrôle en fournissant l'adresse de votre propre routine.

L'utilisation de ces fonctions nécessite bien entendu prudence et rigueur.

Notez qu'à l'entrée de votre routine, le système a déjà masqué les interruptions mais n'a pas sauvegardé ses registres.

vex_timv fonction 118

Modifie l'adresse d'interruption timer.

vex_butv fonction 125

Modifie l'adresse des interruptions boutons de souris. Exécutées lorsqu'un bouton de la souris a été pressé.

vex_motv fonction 126

Modifie le branchement de la gestion d'affichage de la souris. Exécutée à chaque déplacement de celle-ci.

vex_curv fonction 127

Modifie l'adresse de la routine de retour d'affichage souris. Exécutée lorsque la souris a été tracée.

Ces quatre fonctions utilisent les mêmes arguments :

```
int id_vdi, *nouv_adr, *anc_adr;

"nom de la fonction" (id_vdi, nouv_adr, anc_adr);
```

Paramètres d'appel :

id_vdi (contrl[6]) : identificateur VDI.
nouv_adr (contrl[7 et 8]) : adresse de la nouvelle routine.

Paramètres de retour :

anc_adr (contrl[9 et 10]) : ancienne adresse de la routine

Note : vex_timv retourne dans intout [0] l'intervalle d'interruption en millisecondes.

Les fonctions AES

Lorsqu'une application GEM souhaite utiliser les fonctions AES, elle se fait identifier par l'ordonnanceur AES qui lui fournit un numéro spécifique (gl_apid), libère l'espace mémoire requis pour sa gestion, et initialise les paramètres suivants (15 mots) :

global (0) : un mot spécifiant la version AES utilisée. [0101].
global (1) : un mot précisant le nombre d'application que cette version de l'AES supporte concurremment. [1] (bien que plusieurs puissent être sur la liste d'attente).
global (2) : le numéro d'application (gl_apid).
global (3) et (4) : un long mot réservé à l'application.
global (5) et (6) : un long mot qui contiendra, après l'appel de la fonction rsrc_load, l'adresse de départ de l'arbre ressource.
global (7) et (8) : réservé.
global (9) et (10) : réservé.
global (11) et (12) : départ des en-têtes d'application. [$38DFA].
global (13) et (14) : réservé.

A chaque appel d'une fonction AES, sont d'autre part initialisés cinq tableaux que nous ne ferons que nommer puisque, contrairement au VDI, le programmeur n'a guère à s'en soucier. Il s'agit de :

param : tableau d'adresses des zones de paramètres.
control : zone de contrôle de la fonction.
int_in : zone des valeurs 16 bits en entrée.
addr_in : zone des valeurs 32 bits (adresses) en entrée.
int_out : zone des valeurs 16 bits en sortie.
addr_out : zone des valeurs 32 bits (adresses) en sortie.

REMARQUES GÉNÉRALES :

Tout au long de l'annexe, lorsqu'un objet ou une fenêtre sont positionnés par leurs coordonnées, celles-ci réfèrent TOUJOURS à leur coin supérieur gauche.

Pour un certain nombre de constantes utilisées par l'AES, ont été indiquées, en lettres majuscules, leurs noms symboliques, tels qu'ils sont reconnus par le fichier GEMDEFS.H du compilateur DIGITAL RESEARCH. Avant de les utiliser, il convient de vérifier, pour tout autre compilateur, leurs authenticités.

Gestion de l'application

appl_init fonction 10

Lance l'application en réservant l'espace mémoire nécessaire aux paramètres internes de l'AES et en utilisant le tableau GLOBAL (voir plus haut).

Retourne l'identificateur de l'application.

Un accessoire doit déclarer gl_apid comme variable externe.

```
int gl_apid;

gl_apid = appl_init ( );
```

Paramètre de retour :

gl_apid (int_out[0]) : identificateur de l'application
si – 1, une erreur existe, l'application ne peut être ouverte.

appl_write fonction 12

Envoie un nombre spécifié d'octets dans un canal de communication permettant le dialogue entre applications (cf appl_read).

```
int ret, ap_idec, nb_oc;
char tamp [nb_oc];

ret = appl_write (ap_id, nb_oc, tamp);
```

Paramètres d'appel :

ap_idec (int_in[0]) : l'identificateur de l'application destinatrice.
nb_oc (int_in[1]) : nombre d'octets à envoyer.
tamp (addr_in[0]) : adresse du tampon contenant le message.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe

appl_read fonction 11

Lit le nombre spécifié d'octets envoyés par une autre application.

```
int ret. ap_idl, nb_oc;
char tamp [nb_oc];

ret = appl_read (ap_idl, nb_oc, tamp);
```

Paramètres d'appel :

ap_idl (int_in[0]) : l'identificateur de l'application réceptrice
nb_oc (int_int[1]) : nombre d'octets à lire.
tamp (addr_in [0]) : adresse du tampon contenant le message

Paramètre de retour :

ret (int_out[0]) : si 0, une errreur existe

appl_find fonction 13

Retourne l'identificateur d'une application dont le nom de fichier est spécifié.

```
int ap_idt;
char nom [9];  /* terminé par la valeur nulle */

ap_idt = appl_find (nom);
```

Paramètre d'appel :

nom (addr_in[0]) : adresse de la chaîne « nom » qui doit obligatoirement comporter 8 caractères (au besoin, complétée par des espaces).

Paramètre de retour :

ap_idt (int_out[0]) : identificateur de l'application recherchée.
Si ap_idt = −1, l'AES n'a rien trouvé.

appl_trecord fonction 15

Enregistre dans un tampon un nombre donné d'événements en provenance de l'utilisateur.

Un bogue, dans la routine AES qui doit traduire l'événement en un code spécifique (cf ci-dessous), rend la fonction difficilement utilisable puisqu'est stockée, à la place du code, l'adresse de la routine qui a géré l'événement.

Espérant qu'une correction sera effectuée avant l'implantation du système en ROM, nous vous donnons ici les valeurs telles qu'elles devraient être enregistrées.

Chaque événement est stocké sur *deux longs mots.*
(HUIT octets et non six comme dans le GEM d'IBM)

Le premier long mot représente(ra) la nature de l'événement :
0 = timer, 1 = bouton, 2 = souris, 3 = clavier.

Le deuxième long mot dépend de l'événement :
si timer, le nombre de millisec. écoulées (en multiple de 20 m/sec).
si bouton, mot faible = état des boutons (cf evnt_mouse)
mot fort = nombre de cliquages
si souris, mot faible = coord. X, mot fort = coord. Y.
si clavier, octet faible du mot fort = code ASCII
octet fort du mot fort = code clavier (trap 1 fonction 1)

```
int nb_ev, ret_ev;
long adr_ev;      /* nombre d'octets necessaires = nb_ev * 8  */

ret_ev = appl_trecord (adr_ev, nb_ev);
```

Paramètre d'appel :

nb_ev (int_in[0]) : nombre d'événements à stocker.
adr_ev (addr_in[0]) : adresse où seront stockés les événements.

Paramètre de retour : ret_ev (int_out [0]) : nombre d'événements stockés

appl_tplay fonction 14

Exécute, à la vitesse spécifiée, les événements enregistrés par appl_trecord.

```
int nb_ev, vit;
long adr_ev;

appl_tplay (adr_ev, nb_ev, vit);
```

Paramètres d'appel :

nb_ev (int_in[0]) : nombres d'événements à exécuter.
vit (int_in[1]) : une valeur entre 1 et 10 000 déterminant la vitesse d'exécution par rapport à l'enregistrement.
50 = vitesse / 2, 100 = idem, 200 = vitesse * 2, etc...
adr_ev (addr_in[0]) : adresse de stockage de l'enregistrement.

appl_exit fonction 19

Désinitialise l'application courante, libère l'espace mémoire réservé par la fonction appl_init.

```
int ret;

ret = appl_exit ( );
```

Paramètre de retour : ret (int_out[0]) : si 0, une erreur existe.

Gestion des événements

Lors de l'attente d'un événement, le retour à l'application ne s'opère que lorsque la condition de sortie a été réalisée.

evnt_keybd fonction 20

Attend qu'une touche du clavier soit pressée.
Les touches SHIFT, CONTROL et ALTERNATE ne provoquent pas à elles seules la sortie de la fonction, mais leur état peut être testé par la fonction graf_mkstate.

```
int ev_clret;

ev_clret = evnt_keybd ( );
```

Paramètre de retour :

ev_clret (int_out[0]) : octet faible = code ASCII de la touche appuyée
octet fort = code de la touche

evnt_button fonction 21

Attend qu'un bouton de la souris soit dans l'état spécifié.

```
int nb_clic, masque, etat_be;
int clic_s, sourx, soury, etat_bs, etat_cl;

clic_s = evnt_button (nb_clic, masque, etat_be,
                      &sourx, &soury, &etat_bs, &etat_cl);
```

Paramètres d'appel :

nb_clic (int_in[0]) : nombre de cliquages attendus.
Permet de définir le nombre de fois qu'un bouton doit entrer dans son état de sortie (et ce durant le laps de temps spécifié par evnt_dclick), avant que le retour de la fonction ne s'effectue.

masque (int_in[1]) : le(s) bouton(s) à tester (en théorie 16).
Sachant que le bit d'ordre 0 correspond au bouton le plus à gauche de la souris, on détermine le ou les boutons de sortie en mettant à 1 le(s) bit(s) leur correspondant. Tout bit mis à 0 masque l'état du bouton qui lui est dévolu et le rend invalide.
Ex : $0001 ne laisse « répondre » que le bouton gauche de la souris.

état_be (int_in[2]) : l'état du bouton qui doit provoquer le retour :
bit correspondant = 0 → retour lorsque appui du bouton.
bit correspondant = 1 → retour lorsque relâchement du bouton.

Paramètres de retour :

clic_s (int_ out[0]) : nombre de cliquages qui ont provoqué le retour.

Valeur comprise entre 1 et valeur spécifiée dans nb_clic.
sourx (int_out[1]) : coord. X de la souris au retour.
soury (int_out[2]) : coord. Y de la souris au retour.
etat_bs (int_out[3]) : état du bouton qui a occasionné le retour.
Si bit à 1 = bouton pressé, bit à 0 sinon.
etat_cl (int_out[4]) : état des touches spéciales du clavier.
Si le bit correspondant est à 1, la touche était enfoncée :
bit 0 = SHIFT (droit)
bit 1 = SHIFT (gauche)
bit 3 = CONTROL
bit 4 = ALTERNATE

evnt_mouse fonction 22

Attend que la souris entre ou sorte d'une zone rectangulaire spécifiée.

```
in drap, x_in, y_in, larg, haut ;
int sourx, soury, état_bs, état_cl ;

evnt_mouse (drap, x_in, y_in, larg, haut,
    &sourx, &soury, &état_bs, &état_cl) ;
```

Paramètres d'appel :

drap (int_in[0]) : drapeau précisant à l'AES la raison du retour.
Si drap = 0, retour de fonction si la souris entre dans rectangle.
Si drap = 1, retour de fonction si la souris sort du rectangle.
x_in (int_in[1]) : coord. X du rectangle
y_in (int_in [2]) : coord. Y du rectangle.
larg (int_in[3]) : largeur du rectangle en pixels.
haut (int_in[4]) : hauteur du rectangle en pixels.

Paramètres de retour :

sourx (int_out[1]) : coord. X du pointeur de souris au retour.
soury (int_out[2]) : coord. Y du pointeur de souris au retour.
état_bs (int_out[3]) : état des boutons de la souris au retour.
Voir fonction précédente.
état_cl (int_[4]) : état des touches spéciales du clavier.
Voir fonction précédente.

evnt_mesag fonction 23

Attente d'un message envoyé par l'AES dans un tampon spécifié.
Le tampon, dont l'adresse est envoyée à la fonction, doit avoir une taille minimum de huit mots (soit seize octets).

Quelque soit le message envoyé par l'AES, les trois premiers mots réfèrent toujours au même contenu :

mot 0 : numéro identifiant le type de message.
mot 1 : identificateur AES de l'application qui envoie le message.
mot 2 : la longueur du message sans inclure les seize premiers octets. Si son contenu est nul, cela signifie que le message envoyé tient dans les huit mots. Dans le cas contraire, un appel à la fonction **appl_read** fournira le reste du message.

Voici la liste des différents messages envoyés par l'AES, leurs significations, ainsi que le contenu spécifique du tampon pour chaque cas :

* indique qu'un item du menu a été sélectionné.
mot 0 : 10 MN_SELECTED
mot 3 : L'index d'objet du titre du menu sélectionné.
mot 4 : L'index d'objet de l'item du menu sélectionné

*Indique un cliquage dans une fenêtre inactive.
mot 0 : 21 WM_TOPPED
mot 3 : l'identificateur de la fenêtre à activer.

*Indique un cliquage dans la case de fermeture de fenêtre.
mot 0 : 22 WM_CLOSED
mot 3 : l'identificateur de la fenêtre à fermer.

*Indique un cliquage dans la case de pleine-fenêtre spécifiant le plus grand élargissement possible de la fenêtre ou, si c'est déjà le cas, le retour à sa taille antérieure.
mot 0 : 23 WM_FULLED
mot 3 : l'identificateur de la fenêtre.

*Indique un cliquage soit sur les flèches blanches de la barre de défilement, soit sur cette barre elle-même, spécifiant une demande de déplacement de la partie visible de la fenêtre.
mot 0 : 24 WM_ARROWED
mot 3 : l'identificateur de fenêtre.
mot 4 : l'action requise selon la localisation du cliquage.

0 = partie grisée verticale → page supérieure demandée
1 = partie grisée verticale → page inférieure demandée
2 = flèche verticale sup. → rangée supérieure
3 = flèche verticale inf. → rangée inférieure
4 = partie grisée horizontale → page de gauche
5 = partie grisée horizontale → page de droite
6 = flèche horizontale gauche → colonne gauche
7 = flèche horizontale droite → colonne droite

* Indique que l'utilisateur a déplacé l'ascenseur horizontal. Le message est retourné lorsque le bouton de souris est relâché.
mot 0 : 25 WM_HSLID
mot 3 : l'identificateur de fenêtre
mot 4 : une valeur comprise entre 0 (position la plus à gauche) et 1 000 (position la plus à droite), précisant la position requise.

* Déplacement de l'ascenseur vertical requis par l'utilisateur.
mot 0 : 26 WM_VSLID
mot 3 : l'identificateur de la fenêtre
mot 4 : une valeur comprise entre 0 (position la plus haute) et 1 000 (position la plus basse), précisant la position requise.

* Indique que l'utilisateur a modifié la taille de la fenêtre. Le message est envoyé au relâchement du bouton de la souris.
mot 0 : 27 WM_SIZED
mot 3 : l'indentification de la fenêtre.
mot 4 : coord. X de la fenêtre (en théorie, inchangée).
mot 5 : coord. Y de la fenêtre (en théorie, inchangée).
mot 6 : largeur de fenêtre requise.
mot 7 : hauteur de fenêtre requise.

* Indique que l'utilisateur a déplacé la fenêtre.
mot 0 : 28 WM_MOVED
mot 3 : l'identificateur de la fenêtre déplacée.
mot 4 : la nouvelle coordonnée X requise.
mot 5 : la nouvelle coordonnée Y requise.
mot 6 : largeur de la fenêtre (en théorie, inchangée).
mot 7 : hauteur de la fenêtre (en théorie, inchangée).

*L'AES indique à l'application que sa fenêtre doit être activée.
mot 0 : 29 WM_NEWTOP
mot 3 : l'identificateur de la fenêtre à activer.

*Indique qu'un retraçage de la zone spécifiée est nécessaire.
mot 0 : 20 WM_REDRAW
mot 3 : l'identificateur de la fenêtre à retracer.
mot 4 : coordonnée X de la portion d'écran à retracer.
mot 5 : coordonnée Y de la portion d'écran à retracer.
mot 6 : largeur de cette même portion.
mot 7 : hauteur de cette même portion.

L'ordre de retraçage est envoyé après l'un des événements suivants :

— un FULL_SIZE s'il implique un agrandissement de la fenêtre.
— un WM_SIZED si taille demandée est plus grande que la précédente.

— à chaque fois qu'un objet (forme dialogue, etc...) a effacé tout ou partie de votre fenêtre.

* Indique à un accessoire que l'item du menu « Bureau » le désignant a été sélectionné.

mot 0 : 40 AC_OPEN

mot 4 : l'identificateur de l'item du menu (cf menu_register).

* Indique à un accessoire que l'AES exige sa fermeture.

mot 0 : 41 AC_CLOSE

mot 3 : l'identificateur de l'item du menu (cf. menu_register)

Le message AC_CLOSE est envoyé par l'AES dans les trois cas suivants :

— l'application courante est terminée.
— un effacement d'écran est demandé.
— la structure de fenêtrage est réinitialisée.

```
int tampon [8];

evnt_mesag (tampon);
```

Paramètre d'appel :

tampon (addr_in[0]) : adresse du tampon recevant les messages.

evnt_timer fonction 24

Attente d'un événement de l'horloge interne selon un délai spécifié.

L'intervalle de temps qui s'écoule entre l'appel et le retour de fonction est précisé dans deux mots.

```
int bas_del, haut_del;

evnt_timer (bas_del, haut_del);
```

Paramètres d'appel :

bas_del (int_in[0]) : valeur basse du délai en millisecondes.

haut_del (int_in[1]) : valeur haute du délai en millisecondes.

evnt-multi fonction 25

Attente simultanée de plusieurs événements.

Les événements attendus peuvent comprendre tout ou partir de ceux définis précédemment (souris, bouton, message, clavier, timer).

Les composantes de cette fonction sont en grande partie celles déjà employées pour les événements particuliers, la seule « nouveauté » consistant en la possibilité de définir deux événements-souris, correspondant chacun à l'entrée ou à la sortie d'une zone rectangulaire spécifique (MU_M1 et MU_M2).

```
int type_ev, nb_clic, masque, etat_be, drap1, x_in1, y_in1;
int larg1, haut1, drap2, x_in2, y_in2, larg2, haut2;
int tampon [8], bas_del, haut_del;

int ev_quel, sourx, soury, etat_bs, etat_cl, ev_clret, clic_s;

ev_quel = evnt_multi (type_ev, nb_clic, masque, etat_be, drap1,
            x_in1, y_in1, larg1, haut1, drap2, x_in2, y_in2,larg2,
            haut2, tampon, bas_del, haut_del, &sourx, &soury,
            &etat_bs, &etat_cl, &etat_clret, &clic_s);
```

Paramètres d'appel :

type_ev (int in[0]) : type des événements attendus.

Chaque bit de cette variable mis à 1 spécifie les événements qui seront pris en compte par la fonction :

0x0001 MU_KEYBD = attente clavier.
0x0002 MU_BUTTON = attente du bouton spécifié.
0x0004 MU_M1 = attente de la souris par rapport au rectangle 1.
0x0008 MU_M2 = attente de la souris par rapport au rectangle 2.
0x0010 MU_MESAG = attente d'un message dans le tampon.
0x0020 MU_TIMER = attente d'un délai d'horloge.

Ex : type_ev = 0x003 ou type_ev = MU_KEYBD | MU_BUTTON indique que la fonction attend un événement clavier ou bouton.

nb_clic (int_in[1])	:	voir evnt_button
masque (int_in[2])	:	
état_be (int_in[3])	:	
drap1 (int_in[4])	:	voir evnt_mouse (MU_M1)
x_in1 (int in[5])	:	
y_in1 (int_in[6])	:	
larg1 (int_in[7])	:	
haut1 (int_in[8])	:	
drap2 (int_in[9])	:	evnt_mouse (MU_M2)
x_in2 (int_in[10])	:	
y_in2 (int_in[11])	:	
larg2 (int_in[12])	:	
haut2 (int_in[13])	:	
bas_del (int_in[14])	:	voir evnt_timer
haut_del (int_in[15])	:	"
tampon (addr_in[0])	:	adresse du tampon de message

Paramètres de sortie :

ev_quel (int_out[0]) : le type d'événement (l'un de ceux demandés dans type_ev) qui a provoqué le retour de la fonction.
sourx (int_out[1]) : coord. X de la souris au retour.
soury (int_out[2]) : coord. Y de la souris au retour.
etat_bs (int_out[3]) : état des boutons (cf. evnt_button).
etat_cla (int_out[4]) : touches spéciales clavier (cf evnt_button)
ev_clret (int_out[5]) : touche clavier appuyée (cf. evnt_keybd).
clic_s (int_ out[6]) : nombre de cliquages (cf. evnt_button).

evnt_dclick fonction 26

Demande ou spécifie l'intervalle de temps entre deux cliquages pour que ceux-ci soient considérés comme double-clic (cf. Panneau de Contrôle dans le Bureau).

```
int vit_dc, drap_dc, sel_dc;

sel_dc = evnt_dclick (vit_dc, drap_dc);
```

Paramètres d'appel :

vit_dc (int_in[0]) : nombre compris entre 0 (le plus lent) et 4.
0 = 450 millisec., 1 = 330, 2 = 275, 3 = 220, 4 = 165.
drap_dc (intin[1]) : drapeau spécifiant s'il s'agit d'une demande (si 0) ou d'une indication à l'AES (si 1).

Paramètre de retour :

sel_dc (int_out[0]) : retourne la vitesse sélectionnée.

Gestion des menus

menu_bar fonction 30

Affiche ou supprime la barre des menus de l'application.

```
int drap_mn, ret;
long adr_mn;

ret = menu_bar (adr_mn, drap_mn);
```

Paramètres d'appel :

drap_mn (int_in [1]) : si 0, retire la barre des menus.
si 1, affiche la barre des menus.
adr_mn (addr_in[0]) : adresse de l'arbre qui définit le menu.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

menu_icheck fonction 31

Affiche ou supprime un « check » (v = caractère de validation) à côté d'un item de menu.

```
int n_item, dp_cit, ret;
long adr_mn;

ret = menu_icheck (adr_mn, n_item, dp_cit);
```

Paramètres d'appel :

n_item (int_in[0]) : index de l'item dans l'arbre-menu.
dp_cit (int_in[1]) : si 0, efface le « check » s'il existe.
si 1, affiche un « check ».
adr_mn (addr_in[0]) : adresse de l'arbre qui définit le menu.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

menu_ienable fonction 32

Active ou désactive un item de menu. S'il est inactif, un item est affiché avec une intensité réduite et sa sélection est impossible.

```
int n_item, dp_ait, ret;
long adr_mn;

ret = menu_ienable (adr_mn, n_item, dp_ait);
```

Paramètres d'appel :

n_item (int_in[0]) : index de l'item dans l'arbre-menu.
dp_ait (int_in[1]) : si 0, désactive l'item de menu.
si 1, active l'item de menu.
adr_mn (addr_in[0]) : adresse de l'arbre qui définit le menu.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

menu_tnormal fonction 33

Affiche un titre de menu en vidéo normale ou inverse.

```
int n_titre, dp_vtit, ret;
long adr_mn;

ret = menu_tnormal (adr_mn, n_titre, dp_vtit);
```

Paramètres d'appel :

n_titre (int_in[0]) : index du titre dans l'arbre-menu.
dp_vtit (int_in[1]) : si 0, affiche en vidéo inverse.
si 1, affiche en vidéo normale.
adr_mn (addr_in[0]) : adresse de l'arbre qui définit le menu.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

menu_text fonction 34

Remplace le texte d'un item de menu.

```
int n_item, ret;
char texte [];  /* chaine termine par un caractere nul */
long adr_mn;

ret = menu_text (adr_mn, n_item, texte);
```

Paramètres d'appel :

n_item (int_in[0]) : index de l'item dans l'arbre-menu.
adr_mn (addr_in[0]) : adresse de l'arbre qui définit le menu.
texte (addr_in[1]) : adresse du nouveau texte (qui ne peut être plus long que l'ancien).

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

menu_register fonction 35

Place le titre d'un accessoire dans le menu « Bureau ».

```
 int gl_apid, n_acit;
 char titre [ ];

n_acit = menu_register (gl_apid, titre);
```

Paramètres d'appel :

gl_apid (int_in[0]) : identificateur de l'accessoire (cf. appl_init).
titre (addr_in[0]) : adresse de la chaîne.

Paramètres de retour :

n_acit (int_out[0]) : index de l'item (de 0 à 5). Si −1, erreur.

Gestion des objets

objc_add fonction 40

Ajoute un objet à un arbre en définissant ses inter-relations.

```
int ob_par, ob_enf, ret;
long adr_arb;

ret = objc_add (adr_arb, ob_par, ob_enf);
```

Paramètres d'appel :

ob_par (int_in[0]) : l'index du parent de l'objet ajoutée.
ob_enf (int_in[1]) : l'index de l'objet lui-même.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

objc_delete fonction 41

Supprime un objet dans un arbre.

```
int ob_sup, ret;
long adr_arb;

ret = objc_delete (adr_arb, ob_sup);
```

Paramètres d'appel :

ob_sup (int_in[0]) : l'index de l'objet à supprimer.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

objc_draw fonction 42

Trace un objet et sa descendance, avec affichage restreint.

```
int ob_deb, ob_desc, x_clip, y_clip, lg_clip, ht_clip, ret;
long adr_arb;

ret = objc_draw (adr_arb, ob_deb, ob_desc, x_clip, y_clip,
                 lg_clip, ht_clip);
```

Paramètres d'appel :

ob_deb (int_in[0]) : l'index du premier objet à tracer.
ob_desc (int_in[1]) : niveau de descendance à tracer.
si = 0, ne dessine que l'objet lui-même.
si = 1, dessine l'objet + enfants de l'objet, etc...
x_clip (int_in[2]) : coord. X du rectangle de restriction (cf vs_clip)
y_clip (int_in[3]) : coord. Y de ce rectangle.
lg_clip (int_in[4]) : largeur du rectangle.
ht_clip (int_in[5]) : hauteur du rectangle.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

objc_find fonction 43

Cherche l'objet qui se trouve sous les coordonnées spécifiées.

```
int ob_rech, rech_lim, x_coor, y_coor, rep;
long adr_arb;

rep = objc_find (adr_arb, ob_rech, rech_lim, x_coor, y_coor);
```

Paramètres d'appel :

ob_rech (int_in[0]) : index de l'objet par qui débute la recherche.
rech_lim (int_in[1]) : niveau de descendance dans laquelle doit s'effectuer la recherche (0, 1, 2, etc... cf fonction précédente).

x_coor (int_in[2]) : coord. X du point de recherche.
y_coor (int_in[3]) : coord. Y du point de recherche.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètre de retour :

rep (int_out[0]) : l'index de l'objet trouvé. Si $-1 \rightarrow$ pas trouvé.

objc_offset fonction 44

Calcule et fournit les coordonnées d'un objet par rapport à l'écran.

```
int ob_quel, x_obj, y_obj, ret;
long adr_arb;

ret = objc_offset (adr_arb, ob_quel, &x_obj, &y_obj);
```

Paramètres d'appel :

ob_quel (int_in[0]) : index de l'objet dont on cherche les coord.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètres de retour :

ret (in_out[0]) : si 0, une erreur existe.
x_obj (in_out[1]) : coord. X de l'objet
y_obj (int_out[2]) : coord. Y de l'objet.

objc_order fonction 45

Modifie l'ordre de disposition des objets.

```
long adr_arb;

ret = objc_order (adr_arb, ob_quel, ob_pos);
```

Paramètres d'appel :

ob_quel (int_in[0]) : l'index de l'objet à déplacer.
ob_pos (int_in[1]) : nouvelle position de l'objet (si $0 \rightarrow$ dernier de la liste, si $1 \rightarrow$ avant-dernier, etc... et si $-1 \rightarrow$ premier de la liste).
adr_arb (addr_in[0]) : l'adresse de l'arbre propriétaire de l'objet.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

objc_edit fonction 46

Permet d'afficher du texte dans un objet de type G_TEXT ou G_BOX-TEXT (cf la structure TEDINFO)

```
int ret. ob_quel, carac. cur_in, aff_ty, cur_out.;
long adr_arb;

ret = objc_edit (adr_obj, ob_quel, carac, &cur_in, aff_ty);
```

A l'adresse de cur_in on placera la position horizontale de départ et on y récupérera la position du curseur au retour.

Paramètres d'appel :

ob_quel (int_in[0]) : index de l'objet.
carac (int_in[1]) : caractère à afficher.
cur_in (int_in[2]) : position du caractère à afficher.
aff_ty (int_in[3]) : type d'affichage demandé (1, 2 ou 3)
1 = fait apparaître le curseur, met à jour les contenus de la chaîne par défaut (te_petext) et du gabarit (te_ptmplt).
2 = affiche le caractère si valide (te_pvalid).
3 = supprime le curseur.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.
cur_out (int_out[1]) : position suivante du curseur.

objc_change fonction 47

Modifie l'état d'un objet et retrace l'objet, si demande en est faite, dans les limites de la zone « clip » (cf vs_clip du VDI).

```
int ob_quel, x_clip, y_clip, lg_clip, ht_clip;
int nv_etat, dp_des, ret;
long adr_arb;

ret = objc_change (adr_arb, ob_quel, 0,x_clip, y_clip, lg_clip,
                   ht_clip, nv_etat, dp_des);
```

Paramètres d'appel :

ob_quel (int_in[0]) : index de l'objet à modifier.
(int_in[1]) : réservé, toujours égal à zéro.
x_clip (int_in[2]) : coord. X du rectangle de « clip »
y_clip (int_in[3]) : coord. Y de ce rectangle
lg_clip (int_in[4]) : largeur du rectangle
ht_clip (int_in[5]) : hauteur du rectangle.

nv_état (int_in[6]) : nouvel état de l'objet (SELECTED, etc...).
dp_des (int_in[7]) : drapeau de retraçage : 0 = ne pas retracer.
1 = retracer.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

Gestion des formes

form_do fonction 50

Laisse l'AES gérer les interactions avec une forme de dialogue. L'application ne reprend le contrôle que lorsqu'une condition de sortie a été sélectionnée par l'utilisateur.

```
int fo_act, fo_ret;
long fo_adr;

fo_ret = form_do (adr_arb, fo_act);
```

Paramètres d'appel :

fo_act (int_in[0]) : l'index objet du champ d'édition à activer à l'entrée dans le dialogue. Si pas de champ, fo_act = 0.
adr_arb (addr_in[0]) : l'adresse de l'arbre propriétaire de la forme.

Paramètres de retour :

fo_ret (int_out[0]) : l'index objet responsable de la sortie du dialogue.

form_dial fonction 51

Dépend de la valeur d'un drapeau et agit comme suit :

fo_dp = 0 FMD_START : réserve un espace écran pour afficher une boîte dialogue.
fo_dp = 1 FMD_GROW (cf graf_growbox) : génère un effet visuel expansif donnant une illusion d'ouverture.
fo_dp = 2 FMD_SHRINK (cf graf_shrinkbox) : génère un effet visuel rétractif donnant une illusion de fermeture.
fo_dp = 3 FMD_FINISH : libère l'espace écran, génère une demande de retraçage (cf evnt_mesag).

```
int fo_dp, x_pcoor, y_pcoor, l_prect, h_prect;
int x_gcoor, y_gcoor, l_grect, h_grect;

ret = form_dial (fo_dp, x_pcoor, y_pcoor, l_prect, h_prect,
                 x_gcoor, y_gcoor, l_grect, h_grect);
```

Paramètres d'appel :

fo_dp (int_in[0]) : drapeau de fonction (voir ci-dessus).
x_pcoor (int_in[1]) : coord. X de la petite boîte.
Si GROW = source de l'expansion, si SHRINK = desti. de rétraction.
y_pcoor (int_in[2]) : coord. Y de cette même boîte.
l_prect (int_in[3]) : largeur de petite boîte.
h_prect (int_in[4]) : hauteur de petite boîte.
x_gcoor (int_in[5]) : coord. X de la grande boîte.
y_gcoor (int_in[6]) : coord. Y de cette même boîte.
l_grect (int_in[7]) : largeur de grande boîte.
h_grect (int_in[8]) : hauteur de grande boîte.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

NOTE : pour FMD_START et FMD_FINISH, les valeurs de int_in[1] à int_in[4] ne sont pas prises en compte mais demeurent obligatoires pour le bon positionnement des paramètres.

form_alert fonction 52

Affiche une forme d'alerte dont les composants sont spécifiés dans la chaîne. Cette fonction gère elle-même la sauvegarde et la restitution d'écran.

```
int def_bout, bout_sort;
char chaine [];  /* 5 lignes de 30 caractères au maxi

bout_sort = form_alert (def_bout, chaine);
```

Paramètres d'appel :

def_bout (int_in[0]) : bouton par défaut (si touche <RETURN>).
0 = aucun, 1 = premier bouton, 2 = deuxième, 3 = troisième.
chaîne (addr_in[0]) : adresse de la chaîne à afficher.

Paramètres de retour :

bout_sort (int_out[0]) : numéro du bouton sélectionné (1, 2 ou 3).

form_error fonction 53

Affiche une forme d'erreur dont le contenu est déterminé par le code demandé. Même remarque que pour **form_alert.**

```
int bout-sort, code;

bout_sort = form_error (code);
```

Paramètre d'appel : code (int_in[0]) : code de l'erreur.
de − 1 à − 17 = erreurs d'entrée-sortie (BIOS)
de − 32 à − 49 = erreurs disques (DOS).

Paramètre de retour : bout_sort (int_out[0]) : bouton sélectionné.

form_center fonction 54

Retourne les coordonnées, centrées sur l'écran, d'une boîte de dialogue.

```
int x_c, y_c, lg_c, ht_c;
long fo_adr;

form_center (fo_adr, &x_c, &y_c, &lg_c, &ht_c);
```

Paramètre d'appel :

fo_adr (addr_in[0]) : adresse de l'arbre qui définit la forme.

Paramètres de retour :

x_c (int_out[1]) : coordonnée X centrée.
y_c (int_out[2]) : coordonnée Y centrée.
lg_c (int_out[3]) : largeur de la boîte de dialogue.
ht_c (int_out[4]) : hauteur de la boîte de dialogue.

Gestion graphique

graf_rubberbox fonction 70

Trace une boîte en traits pointillés, expansible ou rétractable par l'utilisateur, mais fixe à son coin supérieur gauche.

```
int b_x, b_y, lg_min, ht_min, fin_lg, fin_ht, ret;

ret = graf_rubberbox (b_x, b_y, lg_min, ht_min,
                      &fin_lg, &fin_ht);
```

Paramètres d'appel :

b_x (int_in[0]) : coord. X de la boîte.
b_y (int_in[1]) : coord. Y de la boîte.
lg_min (int_in[2]) : largeur minimum de boîte.
ht_min (int_in[3]) : hauteur minimum de boîte.

Paramètres de retour : (au relâchement du bouton)

ret (in_out[0]) : si 0, une erreur existe.
fin_lg (int_out[1]) : largeur de boîte au retour.
fin_ht (int_out[2]) : hauteur de boîte au retour.

graf_dragbox fonction 71

Trace une boîte en traits pointillés que les mouvements de souris déplacent tant qu'un bouton est pressé.

```
int b_lg, b_ht, b_x, b_y, lim_x, lim_y, lim_lg, lim_ht;
int ret, fin_x, fin_y;

ret = graf_dragbox (b_lg, b_ht, b_x, b_y, lim_x, lim_y,
                    lim_lg, lim_ht, &fin_x, &fin_y);
```

Paramètres d'appel :

b_lg (int_in[0]) : largeur de la boîte à déplacer.
b_ht (int_in[1]) : hauteur de la boîte à déplacer.
b_x (int_in[2]) : coord. X initiale de la boîte.
b_y (int_in[3]) : coord. Y initiale de la boîte.
lim_x (int_in[4]) : coord. X du rectangle de limite, zone dans lequel le déplacement est autorisé.
lim_y (int_in[5]) : coord. y du rectangle de limite.
lim_lg(int_in[6] : largeur du rectangle de limite.
lim_ht (int_in[7]) : hauteur du rectangle de limite.

Paramètres de retour : (au relâchement du bouton)

ret (in_out[0]) : si 0, une erreur existe.
fin_x (int_out[1]) : coord. X finale de la boîte.
fin_y (int_out[2]) : coord. Y finale de la boîte.

graf_movebox fonction 72

Trace une boîte en traits pointillés se déplaçant d'une position source à une position destination. Largeur et hauteur non modifiées.

```
int ret, b_lg, b_ht, sc_x, sc_y, dest_x, dest_y;

ret = graf_movebox (b_lg, b_ht, sc_x, sc_y, dest_x, dest_y);
```

Paramètres d'appel :

b_lg (int_in[0]) : largeur de la boîte.
b_ht (int_in[1]) : hauteur de la boîte.
sc_x (int_in[2]) : coord. X source.
sc_y (int_in[3]) : coord. Y source.
dest_x (int_in[4]) : coord. X destination.
dest_y (int_in[5]) : coord. Y destination.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

graf_growbox fonction 73

graf_shrinkbox fonction 74

Trace une boîte expansive (growbox) ou rétractile (shrinkbox) donnant un effet visuel d'ouverture ou de fermeture (cf form_dial).

```
int ret, x_pcoor, y_pcoor, l_pbt, h_pbt;
int x_gcoor, y_gcoor, l_gbt, h_gbt;

ret = "nom de la fonction" (x_pcoor, y_pcoor, l_pbt, h_pbt,
                            x_gcoor, y_gcoor, l_gbt, h_gbt);
```

Paramètres d'appel :

x_pcoor (int_in[0]) : coord. de la petite boîte.
La source si graf_growbox, la destination si graf_shrinkbox.
y_pcoor (int_in[1]) : coord. Y de la petite boîte.
l_pbt (int_in[2]) : largeur de petite boîte.
h_pbt (int_in[3]) : hauteur de petite boîte.
x_gcoor (int_in[4]) : coord. X de la grande boîte.
La destination si graf_growbox, la source si graf_shrinkbox.
y_gcoor (int_in[5]) : coord. Y de la grande boîte.
l_gbt (in_in[6]) : largeur de grande boîte.
h_gbt (int_in[7]) : hauteur de grande boîte.

Paramètres de retour :

ret (in_out[0]) : si 0, une erreur existe.

graf_watchbox fonction 75

Modifie l'état d'un objet spécifié lorsque la souris entre ou sort de la zone de localisation de celui-ci. L'appel à cette fonction se fait lorsque le bouton de souris est pressé, le retour s'effectuant au relâchement.

```
int rep, ob_quel, etat_in, etat_out;

long adr_arb;

rep = graf_watchbox (adr_arb, ob_quel, etat_in, etat_out);
```

Paramètres d'appel :

int_in[0]) = réservé.
ob_quel (int_in[1]) : index de l'objet dans l'arbre.
état_in (int_in[2]) : état de l'objet si entrée de la souris.
état_out (int_in[3]) : état de l'objet si sortie de la souris.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

Paramètre de retour :

rep (int_out[0]) : position de la souris au relâchement du bouton.
0 = en-dehors de l'objet, 1 = à l'intérieur de l'objet.

graf_slidebox fonction 76

Déplace un objet (une boîte) dans son objet-parent et retourne la position de l'enfant relativement à son parent. Permet en fait de créer son propre ascenseur graphique (cf wind_get WF_HSLIDE et WF_VSLIDE).

L'appel se fait lorsque le bouton de souris est pressé, le retour s'effectuant au relâchement.

```
int rep, ob_par, ob_enf, ver_hor;
long adr_arb;

rep = graf_slidebox (adr_arb, ob_par, ob_enf, ver_hor);
```

Paramètres d'appel :

ob_par (int_in[0]) : index de l'objet parent (la cage de l'ascenseur).
ob_enf (int_in[1]) : index de l'objet enfant (l'ascenseur).
ver_hor (int_in[3]) : si 0, ascenseur horizontal.
si 1, ascenseur vertical.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire des 2 objets.

Paramètre de retour :

rep (int_out[0]) : valeur entre 0 et 1 000 (cf wind_get).

graf_handle fonction 77

Retourne l'identificateur de la station de travail (numéro de station ouverte par le Bureau GEM) ainsi que des paramètres sur les caractères.

```
int id_sta, lg_cel, ht_cel, bt_lg, bt_ht;

id_sta = graf_handle (&lg_cel, &ht_cel, &bt_lg, &bt_ht);
```

Paramètres de retour :

id_sta (in_out[0]) : identificateur de la station.
lg_cel (int_out[1]) : largeur de la cellule de caractère.
ht_cel (int_out[2]) : hauteur de la cellule de caractère.
bt_lg (int_out[3]) : largeur d'un carré capable d'encadrer un caractère
bt_ht (int_out[4]) : hauteur de ce même carré.

graf_mouse fonction 78

Selon le drapeau spécifié, visualise, cache ou sélectionne une nouvelle forme de souris.

```
int ret, dp_sour, sour_for [37];

ret = graf_mouse (dp_sour, sour_for);
```

Paramètres d'appel :

dp_sour (int_in[0]) : drapeau spécifiant l'action demandée.
0 = flèche, 1 = barre verticale, 2 = abeille,
3 = main doigt pointé, 4 = main ouverte, 5 = croix fine,
6 = croix épaisse, 7 = croix encadrée.
255 = souris utilisateur (forme stockée dans sour_for).
256 = cache la souris.
257 = visualise la souris dans sa forme courante.
sour_for (addr_in[0]) : adresse du tableau définissant la forme utilisateur (cf vsc_form dans le VDI).

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

graf_mkstate fonction 79

Retourne la position de la souris, l'état des boutons et des touches SHIFT, CONTROL et ALTERNATE (cf evnt_button).

```
int sourx, soury, etat_bs, etat_cl;

graf_mkstate (&sourx, &soury, &etat_bs, &etat_cl);
```

Paramètres de retour :

sourx (int_out[1]) : coord. X de la souris.
soury (int_out[2]) : coord. Y de la souris.
etat_bs (int_out[3]) : état des boutons de souris.
etat_cl (int_out[4]) : état des touches du clavier.

Gestion des fenêtres

wind_create fonction 100

Alloue à l'application la taille de fenêtre spécifiée en entrée, retourne un identificateur de fenêtre.

La taille demandée à la fonction sera la taille maximum de fenêtre (l'écran plein moins la barre des menus, valeurs retournées par **wind_get**) si l'on veut donner à l'utilisateur une possibilité d'agrandissement maximum.

Au retour, la fenêtre n'est pas ouverte (cf wind_open).

```
int id_fen, comp_fen, x_fen, y_fen, lg_fen, ht_fen;

id_fen = wind_create (comp_fen, x_fen, y_fen, lg_fen, ht_fen);
```

Paramètres d'appel :

comp_fen (int_in[0]) : valeur précisant les composants de la fenêtre.
Si le bit correspondant est mis à 1, le composant existe :

bit 0 :	= NAME	(0x0001) : titre de fenêtre.
bit 1 :	= CLOSER	(0x0002) : case de fermeture.
bit 2 :	= FULLER	(0x0004) : case de pleine-fenêtre.
bit 3 :	= MOVER	(0x0008) : barre de déplacement.
bit 4 :	= INFO	(0x0010) : ligne d'information.
bit 5 :	= SIZER	(0x0020) : case de dimensionnement.
bit 6 :	= UPPARROW	(0x0040) : flèche supérieure.
bit 7 :	= DNARROW	(0x0080) : flèche inférieure.
bit 8 :	= VSLIDE	(0x0100) : ascenseur vertical.
bit 9 :	= LFARROW	(0x0200) : flèche gauche.
bit 10 :	= RTARROW	(0x0400) : flèche droite.
bit 11 :	= HSLIDE	(0x0800) : ascenseur horizontal.

x_fen (int_in[1]) : coord. X de la fenêtre.

y_fen (int_in[2]) : coord. Y de la fenêtre.
lg_fen (int_in[3]) : largeur de la fenêtre.
ht_fen (int_in[4]) : hauteur de la fenêtre.

Paramètre de retour :

id_fen (int_out[0]) : identificateur de fenêtre (de 1 à 8, 0 étant l'identificateur de la fenêtre du bureau GEM).
Si id_fen négatif → l'AES n'a plus de fenêtre en réserve.

wind_open fonction 101

Ouvre une fenêtre aux coordonnées et à la taille spécifiée, fenêtre qui doit au préalable avoir été créée (cf wind_create).

```
int ret, id_fen, x_fen, y_fen, lg_fen, ht_fen;

ret = wind_open (id_fen, x_fen, y_fen, lg_fen, ht_fen);
```

Paramètres en entrée :

id_fen (int_in[0]) : identificateur de la fenêtre à ouvrir.
x_fen (int_in[1]) : coord. X de la fenêtre.
y_fen (int_in[2]) : coord. Y de la fenêtre.
lg_fen (int_in[3]) : largeur de la fenêtre.
ht_fen (int_in[4]) : hauteur de la fenêtre.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

wind_get fonction 104

Retourne jusqu'à quatre information spécifiques, dont le contenu dépend de la valeur d'un drapeau.

* drapeau = 4 WF_WXYWH
Les coord. de la zone de travail courante (fenêtre moins bordures).
* drapeau = 5 WF_CXYWH
Les coord. de la fenêtre entière courante (bordures incluses).
* drapeau = 6 WF_PXYWH
Les coord. précédentes de la fenêtre (bordures incluses).
* drapeau = 7 WF_FXYWH
Les coord. de la plus grande fenêtre possible (bordures incluses).
* drapeau = 11 WF_FIRSTXYWH
Les coord. du premier rectangle de retraçage

* drapeau = 12 WF_NEXTXYWH
Les coord. du rectangle de retraçage suivant.

Pour ces six valeurs de drapeau, sont retournés :
arg1 = coord. X arg2 = coord. Y
arg3 = largeur fenêtre arg4 = hauteur fenêtre.

* drapeau = 8 WF_HSLIDE
Une valeur entre 1 et 1 000 donnant la position de l'ascenseur horizontal.
arg1 = de 1 (position gauche extrême) à 1 000 (position droite extrême).
* drapeau = 9 WF_VSLIDE
Une valeur entre 1 et 1 000 donnant la position de l'ascenseur vertical.
arg1 = de 1 (position haute extrême) à 1 000 (position basse extrême).
* drapeau = 10 WF_TOP
L'identificateur de la fenêtre actuellement active.
arg1 = identificateur de la fenêtre.
* drapeau = 15 WF_HSLISIZ
Une valeur entre 1 et 1 000 donnant la taille de l'ascenseur horizontal relativement à la barre de défilement.
arg1 = de 1 à 1 000 (si – 1, taille minimum).
* drapeau = 16 WF_VSLSIZ
Identique à drapeau = 15, mais pour l'ascenseur vertical.

```
int ret, id_fen, drap, arg1, arg2, arg3, arg4;

ret = wind_get (id_fen, drap, &arg1, &arg2, &arg3, &arg4);
```

Paramètres d'appel :

id_fen (int_in[0]) : identificateur de la fenêtre.
drap (int_in[1]) : drapeau spécifiant le type d'informations.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.
arg1 (int_out[1]) : dépendant de la valeur du drapeau.
arg2 (int_out[2]) : dépendant de la valeur du drapeau.
arg3 (int_out[3]) : dépendant de la valeur du dreapeau.
arg4 (int_int[4] : dépendant de la valeur du drapeau.

wind_set fonction 105

Modifie un ou des contenus spécifiques, selon la valeur d'un drapeau.

* drapeau = 1 WF_KIND
Les composants de bordure de la fenêtre.
arg1 = valeur résultante des composants (cf wind_create).

* drapeau = 2 WF_NAME
L'adresse de la chaîne contenant le nom de la fenêtre.
arg1 et arg2 contiennent l'adresse.
* drapeau = 3 WF_INFO
L'adresse de la chaîne contenant la ligne d'information.
arg1 et arg2 contiennent l'adresse.
* drapeau = 5 (cf wind_get).
* drapeau = 8 (cf wind_get).
* drapeau = 9 (cf wind_get).
* drapeau = 10 (cf wind_get).
* drapeau = 14 WF_NEWDESK
L'adresse de l'arbre d'objets remplaçant l'arbre du Bureau GEM.
arg1 = mot faible de l'adresse de l'arbre.
arg2 = mot fort de l'adresse de l'arbre.
arg3 = index du premier objet.
* drapeau = 15 (cf wind_get).
* drapeau = 16 (cf wind_get)

```
int ret, id_fen, drap, arg1, arg2, arg3, arg4;

ret = wind_get (id_fen, drap, arg1, arg2, arg3, arg4);
```

Paramètres d'appel :

id_fen (int_in[0]) : identificateur de fenêtre.
drap (int_in[1]) : drapeau précisant la modification.
arg1 (int_in[2]) : dépendant de la valeur du drapeau.
arg2 (int_in[3]) : dépendant de la valeur du drapeau.
arg3 (int_in[4]) : dépendant de la valeur du drapeau.
arg4 (int_in[5]) : dépendant de la valeur du drapeau.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

wind_find fonction 106

Retourne l'identificateur de la fenêtre située sous la position spécifiée.

```
int id_quel, sourx, soury;

id_quel = wind_find (sourx, soury);
```

Paramètres d'appel :

sourx (int_in|0|) : coord. X du point (souvent relatif à la souris).
soury (int_in|1|) : coord. Y du point.

Paramètre de retour :

id_quel (int_out|0|) : identificateur de la fenêtre sous le point.

wind_update fonction 107

Selon la valeur d'un drapeau, indique à l'AES que :

* drapeau = 0 END_UPDATE

L'application a terminé son retraçage de fenêtre, elle débloque la liste des rectangles de retraçage (annule l'effet de BEG_ UPDATE).

* drapeau = 1 BEG_UPDATE

L'application débute une opération de retraçage, elle gèle la liste des rectangles de retraçage interdisant ainsi toute interférence.

* drapeau = 2 END_MCTRL

Rend le contrôle de la souris à l'AES (annule l'effet de BEG_MCTRL).

* drapeau = 3 BEG_MCTRL

L'application prend le contrôle total de la souris : l'AES ne répond plus des barres de menus et de commandes.

```
int ret, drap;

ret = wind_update (drap);
```

Paramètre d'appel : drap (int_in|0|) : drapeau de la fonction.

Paramètre de retour : ret (int_out|0|) : si 0, une erreur existe.

wind_calc fonction 108

Retourne, d'après les composants de la fenêtre, et selon la valeur d'un drapeau :

* drapeau = 0 WC_BORDER

Les coordonnées de la fenêtre totale (bordures incluses) à partir des paramètres d'appel définissant la zone de travail.

* drapeau = 1 WC_WORK

Les coordonnées de la zone de travail (bordures non incluses) à partir des paramètres d'appel définissant la fenêtre totale.

```
int drap, comp_fen, in_x, in_y, in_lg, in_ht;
int ret, out_x, out_y, out_lg, out_ht;

ret = wind_calc (drap, comp_fen, in_x, in_y, in_lg, in_ht,
                 &out-x, &out_y, &out_lg, &out_ht);
```

Paramètres d'appel :

drap (int_in[0]) : valeur du drapeau (0 ou 1).
comp_fen (int_in[1]) : composants de la fenêtre (cf wind_create).
in_x (int_in[2]) : coord. X d'appel.
in_y (int_in[3]) : coord. Y d'appel
in_lg (int_in[4]) : largeur d'appel.
in_ht (int_in[5]) : hauteur d'appel.

Paramètres de retour :

ret (int_out[0]) : Si 0, une erreur existe.
out_x (int_out[1]) : coord. X de retour.
out_y (int_out[2]) : coord. Y de retour
out_lg (int_out[3]) : largeur de retour.
out ht (int_out[4]) : hauteur de retour.

wind_close fonction 102

Ferme une fenêtre (sans la supprimer : cf wind_delete).

```
int ret, id_fen;

ret = wind_close (id_fen);
```

Paramètre d'appel : id_fen (int_in[0]) : identif. de la fen. à fermer.

Paramètre de retour : ret (int_out[0]) : si 0, une erreur existe.

wind_delete fonction 103

Supprime une fenêtre préalablement fermée, libère son identificateur.

```
int ret, id_fen;

ret = wind_delete (id_fen);
```

Paramètre d'appel : id-fen (int_in[0]) : identif. de la fen. à supprimer.

Paramètre de retour : ret (int_out[0]) : si 0, une erreur existe.

rsrc_load fonction 110

Charge un fichier ressource du disque vers la mémoire.

```
int ret;
char nom [ ];

ret = rsrc_load (nom);
```

Paramètre d'appel :

nom (addr_in|0|) : adresse de la chaîne « nom du fichier ».

Paramètre de retour :

ret (int_out|0|) : si 0, une erreur existe.

rsrc_gaddr fonction 112

Fournit l'adresse d'une structure de données soit chargées par **rsrc_load**, soit stockée par **rsrc_saddr.**

```
int type, index, ret;
long adr_rsc;

ret = rsrc_gaddr (type, index, &adr_rsc);
```

Paramètres d'appel :

type (int_in|0|) : valeur de 0 à 16, définissant le type de structure dont on désire l'adresse.

0	R_TREE	= structure d'arbre.
1	R_OBJET	= structure d'objet.
2	R_TEDINFO	= structure d'édition.
3	R_ICONBLK	= structure d'icone.
4	R_BITBLK	= structure de bits.
5	R_STRING	= pointeur de chaîne.
6	R_IMAGEDATA	= pointeur d'image.
7	R_OBSPEC	= pointeur dans une structure OBJECT.
8	R_TEPTEXT	= pointeur dans une structure TEDINFO
9	R_TEPTMPLT	= pointeur dans une structure TEDINFO.
10	R_TEPVALID	= pointeur dans une structure TEDINFO.
11	R_IBPMASK	= pointeur dans une structure ICONBLK.
12	R_IBPDATA	= pointeur dans une structure ICONBLK.

13 R_IBPTEXT = pointeur dans une structure ICONBLK.
14 R_BIPDATA = pointeur dans une structure BITBLK.
15 R_FRSTR = adresse d'un pointeur de chaîne.
16 R_FRIMG = adresse d'un pointeur d'image.
index (int_in[1]) : index de l'objet.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.
adr_rsc (addr_out[0]) : adresse de la structure désirée.

rsrc_saddr fonction 113

Stocke l'adresse d'une structure de données en mémoire.

```
int ret, type, index;
long adr_rsc.

ret = rsrc_saddr (type, index, adr_rsc);
```

Paramètres d'appel :

type (int_in[0]) : valeur de 0 à 16 (cf. rsrc_gaddr).
index (int_in[1]) : index de l'objet.
adr_rsc (addr_in[0]) : adresse de stockage de la structure.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

rsrc_obfix fonction 114

Convertir les coordonnées et la taille d'un objet, d'une référence de taille de caractères en référence de pixels.

```
int index;
long adr_arb;

rsrc_obfix (adr_arb, index);
```

Paramètres d'appel :

index (int_in[0]) : index de l'objet à transformer.
adr_arb (addr_in[0]) : adresse de l'arbre propriétaire de l'objet.

rsrc_free fonction 111

Libère l'espace mémoire alloué par rsrc_load.

```
int ret;

ret = rsrc_free ( );
```

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

Gestion des communications

shel_read fonction 120

Permet de lire les commandes fournies par l'utilisateur au lancement de l'application.

```
char cmd_tp [128], arg_tp [128];

shel_read (cmd_tp, arg_tp);
```

Paramètres d'appel :

cmd_tp (addr_in[0]) : adresse du tampon qui contiendra la commande. (nom du disque, nom du fichier, etc...).

arg_tp (addr_in[1]) : adresse du tampon qui contiendra les arguments de la commande.

shel_write fonction 121

Enchaîne une nouvelle application.

```
int dp_suite, dp_graf, dp_gem;
char cmd_tp [128], arg_tp [128];

shel_write (dp_suite, dp_graf, dp_gem, cmd_tp, arg_tp);
```

Paramètres d'appel :

dp_suite (int_in[0]) : drapeau de chaînage.
0 = retour au système, 1 = chaînage d'application.

dp_graf (int_in[1]) : drapeau graphique.
0 = l'application à chaîner n'est pas graphique, 1 = elle l'est.

dp_gem (int_in[2]) : drapeau GEM.
0 = l'application à chaîner n'est pas GEM, 1 = elle l'est.

cmd_tp (addr_in[0]) : adresse du tampon de la commande à exécuter.

arg_tp (addr_in[1]) : adresse du tampon des arguments de la commande.

shel_find fonction 124

Lit sur disque et retourne des informations sur le fichier spécifié (cf trap 1, fonction $4E), 5 étant l'attribut de recherche de la fonction (cf trap 1, fonction $43).

```
int ret;
char sh_tp [64];

ret = shel_find (sh_tp);
```

Paramètre d'appel :

sh_tp (addr_in[0]) : adresse du tampon qui contient
en entrée : le nom du fichier cherché.
au retour : le bloc d'informations sur le fichier.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

shel_envrn fonction 125

Cherche une chaîne dans le tampon-système de l'AES.

```
char chaine [];
long adr_car;

 shel_envr (adr_car, chaine);
```

Paramètres d'appel :

adr_car (addr_in[0]) : adresse où sera stockée par la routine l'adresse du caractère nul suivant la chaîne trouvée.
chaîne (addr_in[1]) : adresse du nom de la chaîne à chercher.

shel_get fonction 122

Lit en mémoire le fichier DESKTOP.INF, fichier chargé par le système et contenant tous les paramètres d'affichage du Bureau (emplacement des icones, taille et localisation des fenêtres, etc...). C'est ce fichier qui est placé sur disque lors d'une sauvegarde de Bureau.

```
int ret, nb_oc;
char tp_lu [nb_oc];

ret = shel_get (tp_lu, nb_oc);
```

Paramètres d'appel :

nb_oc (int_in[0]) : nombre d'octets à lire.
tp_lu (addr_in[0]) : adresse du tampon de stockage.

Paramètres de retour :

ret (int_out[0]) : si 0, une erreur existe.

shel_put fonction 123

Écrit dans le tampon réservé au fichier DESKTOP.INF (cf ci-dessus), permettant ainsi de modifier la disposition du Bureau lors d'un retour à celui-ci.

```
int ret, nb_oc;
char tp_ecr [nb_oc];

ret = shel_put (tp_ecr, nb_oc);
```

Paramètres d'appel :

nb_oc (int_in[0]) : nombre d'octets à écrire.
tp_ecr (addr_in[0]) : adresse du tampon d'écriture.

Paramètre de retour :

ret (int_out[0]) : si 0, une erreur existe.

Gestion du sélecteur de fichier

L'AES dispose d'une fonction très complète de sélecteur de fichier permettant l'affichage du répertoire d'un disque et la sélection d'un de ses fichiers.

Les arguments fournis par le programmeur définissent l'affichage par défaut ainsi que la sélection par défaut à l'entrée de la fonction.

Mais l'AES n'empêche nullement l'utilisateur de modifier les paramètres du répertoire et donc d'avoir accès à une autre unité de disquette ou à d'autres types de fichiers.

A l'entrée de la fonction, s'ouvre une boîte de dialogue disposant d'une fenêtre d'affichage avec ascenseur graphique, de deux boutons de sortie, et de deux champs d'édition qu'il nous faut détailler.

Le premier (actif à l'ouverture du dialogue) permet à l'utilisateur de taper au clavier le nom du fichier à sélectionner. Si ce dernier existe déjà, cliquer

sur le nom existant place ce nom sur le champ de sélection. Double-cliquer sélectionne directement le fichier sans avoir à valider le bouton de sortie « OK ».

Le deuxième champ (que l'on rend actif en cliquant sur la ligne ou en appuyant sur la flèche haute de direction) correspond au type de répertoire, que l'utilisateur peut modifier en remplaçant le message préexistant (par la commande CPM de son choix) puis en cliquant dans la fenêtre d'affichage.

Exemple : A : *.* affiche tous les fichiers du disque A.
B : *.E?? affiche tout fichier du disque B dont l'appendice commence par la lettre E.

fsel_input fonction 90

Cette fonction nécessite *un minimum de quatre Ko* pour fonctionner : pensez-y lors de vos réservations de mémoire.

L'appel se fait en fournissant deux adresses de chaîne dont le contenu, en ASCII, spécifie les champs d'édition par défaut (si aucun nom de fichier par défaut, la chaîne **tp_fich** est vide). Au retour, ces chaînes contiennent la sélection de l'utilisateur.

```
int ret, bt_sel;
char tp_type [], tp_fich [];

ret = fsel_input (tp_type, tp_fich, &bt_sel);
```

Paramètres d'appel :

tp_type (addr_in|0|) : adresse du type de fichier par défaut.
tp_fich (addr_in|1|) : adresse du nom de fichier par défaut.

Paramètres de retour :

ret (int_out|0|) : si 0, une erreur existe.
bt_sel (int_out|1|) : le bouton sélectionné par l'utilisateur.
0 = annuler, 1 = OK.

Gestion du « presse-papiers »

scrp_write fonction 80

Copie le nom d'un fichier dans un tampon spécifique, non affecté à la sortie de l'application. Permet à différentes applications de se communiquer des fichiers par l'intermédiaire du disque.

```
int ret;
long adr_fich;

ret = scrp_write (adr_fich);
```

Paramètre d'appel : adr_fich (addr_in[0]) : adresse du nom de fichier

Paramètre de retour : ret (int_out[0]) : si 0, une erreur existe

scrp_read fonction 80

Lit le nom du fichier stocké dans un tampon par une autre application (cf scrp_write).

```
int ret;
long adr_fich;

ret = scrp_read (adr_fich);
```

Paramètre d'appel : adr_fich (addr_in[0]) : adresse où doit être stocké le nom contenu dans le tampon

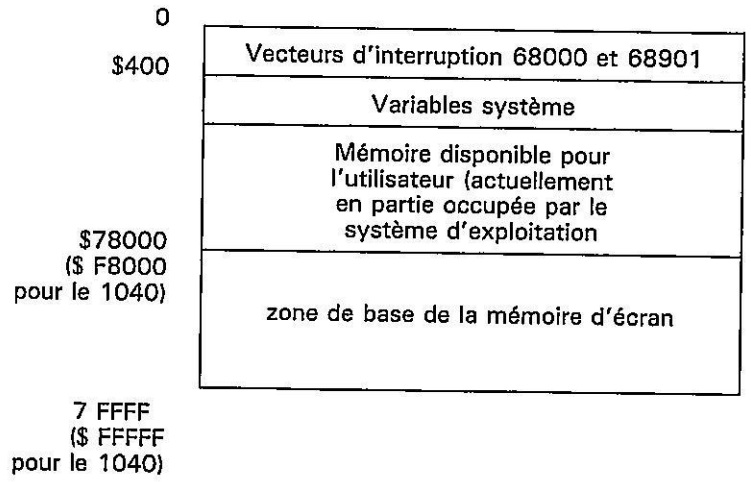
Paramètre de retour : ret (int_out[0]) : si 0, une erreur existe.

ANNEXE 3
CARTE MÉMOIRE

Prétendre établir une carte mémoire détaillée pour les 520 ST serait malhonnête pour plusieurs raisons :

1) Le système d'exploitation, encore résident en mémoire vive à l'heure où est écrit ce livre, devrait être implanté en mémoire morte lorsque ce même ouvrage sera imprimé...
2) Le 68000 possède des modes d'adressage permettant l'allocation dynamique des mémoires et la relogeabilité des programmes. Ces deux possibilités sont exploitées sur le ST et font de la mémoire vive utilisateur une zone en continuelle évolution.

Plusieurs grandes zones peuvent cependant être définies, sinon délimitées, qui autorisent la carte schématique suivante :



La mémoire morte adressable par le ST peut être divisée en deux zones :

de $FA0000 à $FBFFFF : 128 Ko réserves à la cartouche ROM
de $FC0000 à $FEFFFF : 192 Ko, dans lesquels doit être placé le système d'exploitation. D'ores et déjà 64 Ko, de $FC0000 à $FCFFFF, servent au démarrage du ST.

Certaines adresses, dans la zone mémoire s'étendant de $FF8000 à $FFFFFF, correspondent à des circuits périphériques (rappelons que $FFFFFF représente la plus haute mémoire adressable par le 68000). Ces adresses et celles des variables système en début de RAM constituent les seules bases réellement sûres pour un adressage absolu. Ces bases, bien que fort restreintes, ouvrent toutefois des possibilités étendues, justifiant à notre sens l'étude exhaustive ci-dessous.

Variables système

Les adresses allant de 0 à $FF sont réservées aux vecteurs d'interruption 68000 tandis que les adresses allant de $100 à $1FF sont réservées aux vecteurs d'interruption 68901 (cf. chapitre 5). La zone s'étendant de $200 à $3FF est théoriquement réservée à l'utilisateur. En fait, les mémoires $380 à $3EF servent au système de tampon de stockage lors d'une "erreur fatale" (erreur bus, erreur d'adresse...). Les variables système dont l'adresse est garantie immuable dans les versions ultérieures du TOS sont assez peu nombreuses puisqu'elles vont de $400 à $502. Au-delà de cette adresse, d'autres variables sont utilisables (par exemple les tampons d'entrée série ou clavier) mais leur adresse ne peut être connue que par le biais de variables fixes (qui constituent des pointeurs) ou de Traps (voir TRAP 14, fonctions $2, $10 et $22).

Dans la table suivante, figure d'abord l'adresse de la variable (et sa longueur codée .L pour un long mot, .W pour un mot et .B pour un octet) puis sa signification.

$400.L pointeur vers la routine de traitement des interruptions "timer".
$404.L pointeur vers la routine de traitement des erreurs fatales (renvoie – 1 dans DO pour la routine système).
$408L. pointeur vers la routine de fin de programme (RTS pour la routine système).
$40C.L à $41F Cinq pointeurs longs de routines réservés au GEMDOS, inutilisés dans la version actuelle.
$420L. contient le nombre $752019F3 si le contrôle des mémoires s'est bien effectué au Reset du système.

$424.B configuration de l'espace mémoire (0 si 128Ko, 4 si 520 Ko, 5 si 1Mo).

$426.L contient le nombre $31415926 dans les versions du TOS. antérieures au 20/11/85. Contient 0 depuis. Inutilisé.

$42A.L pointeur vers la routine de réinitialisation du système dans les versions du TOS antérieurs au 20/11/85.

$42E.L haut physique de la RAM ($80000 pour un 520 ST).

$432.L bas de la mémoire vive disponible à l'utilisateur.

$436.L haut de la mémoire vive disponible à l'utilisateur ($78000 sur le 520 ST, coïncide avec le début de la zone écran).

$43A.L contient le nombre $237698AA si le contrôle des mémoires s'est bien effectué au Reset (complément de $420.L).

$43E.W drapeau de verrouillage de lecteur (à 1 si verrouillé).

$440.W période de recherche sur disque (0 = 6 ms, 1 = 12 ms, 2 = 2 ms, 3 = 3 ms). La période par défaut est de 3 ms.

$442.W période du "timer", fixée à $14 pour une fréquence d'interruption timer de 50 Hz.

$444.W drapeau de vérification des écritures sur disque (validé à une valeur non nulle). Drapeau validé par défaut.

$446.W numéro du périphérique d'initialisation (0 si disque A).

$448.W mode de sortie vidéo (0 si NTSC, sinon PAL). Inutilisé.

$44A.W inutilisé.

$44C.W résolution vidéo courante (2 si monochrome, 1 si moyenne résolution, 0 si basse résolution).

$44E.L adresse bas mémoire écran.

$452.W drapeau d'autorisation de traitement des interruptions de rafraîchissement d'écran (0=interdit, 1=autorisé).

$454.W nombre de vecteurs de la liste utilisée lors des interruptions fréquence de trame (voir ci-dessous).

$456.L pointeur vers la liste des adresses des routines à exécuter lors d'interruptions fréquence de trame (pointe en principe $4CE).

$45A.L pointeur vers la liste des 16 mots à charger dans la palette des couleurs (cf. 1.1.3). Si nul, palette système valide.

$45E.L pointeur vers une nouvelle base de mémoire d'écran à valider lors de l'interruption fréquence de trame suivante. Si nul, base de la mémoire écran inchangée.

$462.L compteur d'interruptions fréquence de trame prises en compte (incrementé 70 fois par seconde en monochrome, 60 fois avec un moniteur couleurs).

$466.L compteur global d'interruptions rafraîchissement d'écran (y compris celles qui n'ont pas été validées).

$46A.L pointeur vers la routine d'initialisation du disque dur.

$46E.L pointeur vers la routine d'info. sur état du disque dur.

$472.L pointeur vers la routine retournant le descripteur du disque dur.
$476.L pointeur vers la routine de lecture/écriture sur disque dur.
$47A.L pointeur vers la routine d'initialisation du disque dur.
$47E.L pointeur vers la routine retournant le code de changement de disque dur (cf. Trap 13, fonction $9).
$482.W Non nul si le secteur "boot" du disque a réclamé l'initialisation d'un "COMMAND.PRG", nul sinon.
$484.B Attributs du clavier (bit 0 à 1 : touches sonores ; bit 1 à 1 : auto-répétition validée ; bit 2 à 1 : sonnerie validée).
$486.L et $48A.L vecteurs de sauvegarde (inutilisés).
$48E à $49D zone de stockage du tampon de mémoire MD (cf. Trap 13, 0). ($48E.L : 0, $492.L : adresse de départ de la mémoire libre, $496.L : nombre d'octets libres, $49A.L : 0).
$49E.W utilisé pour le tampon MD par le secteur "boot".
$4AO.W comme $49E.W (voir $48E pour tampon MD).
$4A2.L pointeur vers la zone de sauvegarde des registres du 68000 lors de TRAP 13 et TRAP 14.
$4A6.W nombre de lecteurs de disquettes initialisés (0, 1 ou 2).
$4A8.L pointeur vers la routine de sortie écran.
$4AC.W code ASCII de la dernière touche appuyée (pour auto-rep.).
$4B2.L pointeur vers un tampon descripteur du disque A.
$4B6.L pointeur vers un tampon descripteur du disque B.
$4BA.L compteur d'interruptions horloge, incrementé 200 fois par seconde (base du nombre aléatoire).
$4BE.L inutilisé (réservé).
$4C2.L carte mémoire des lecteurs actifs (bit 0 à 1 si disque A actif, bit 1 à 1 si disque B actif, etc).
$4C6.L pointeur vers le tampon général de stockage des données d'un disque (stockage du secteur "boot").
$4CA.L pointeur vers la routine d'auto-exécution (placé par le "boot" secteur). Nul si pas de routine.
$4CE.L à $4EA.L huit adresses de routines exécutables lors d'interruptions rafraîchissement écran (chacune des routines est exécutée successivement. Une adresse nulle constitue la fin de la liste.
$4EE.W drapeau de copie d'écran (1 si en cours, – 1 si interdite).
$4FO.W inutilisé.
$4F2.L pointeur vers l'adresse de départ du TOS (pointe la routine de départ chaud du système).
$4F6.L pointeur vers le bloc d'informations du "shell".
$4FA.L pointeur vers le début de la mémoire disponible.
$4FE.L pointeur vers le début du GEM AES.
$502.L adresse de la routine de copie d'écran.

Les adresses dédiées à des périphériques sont décrites dans les pages qui suivent. Certaines de ces adresses correspondent à des registres ne pouvant qu'être lus, d'autres à des registres pouvant être lus ou écrits (d'où le codage L, R ou R/L). Toute opération sur ces mémoires, court-circuitant le système d'exploitation, doit être effectuée avec un soin extrême, les risques de "planter" le système étant important...

Contrôleur de mémoire :

$FF8001.B indique la taille de la mémoire (0 : 128 Ko, 4 : 512 Ko).

Contrôleur vidéo :

$FF8201.B base haute de la mémoire d'écran (multiple de 64 Ko).
$FF8203.B base basse de la mémoire d'écran (multiple de 256 octets) base de la mémoire d'écran = 65536 * ($FF8201) + 256 * ($FF8203).
$FF820A.B mode de synchronisation vidéo (2 si Pal, 0 sinon).
$FF8240 à $FF825F 16 mots correspondant à la palette de couleurs.
$FF8260.B les bits 0 et 1 définissent la résolution de l'écran (2 si haute, 1 si moyenne, 0 si base).

Les adresses $FF8205, $FF8207, $FF8209, $FF820B, $FF820D, $FF820F et $FF8261 à $FF827F correspondent à des registres du contrôleur vidéo et permettent différentes manipulations de celui-ci. Leur utilisation est déconseillée.

Contrôleur de disque et de DMA (accès direct à la mémoire)

$FF8604.B accès aux données du contrôleur de disque.
$FF8606.B contrôle de mode DMA.
$FF8609.B base haute du DMA.
$FF860B.B base moyenne du DMA.
$FF860D.B base basse du DMA.

Synthétiseur de son :

$FF8800.B sélection de registre du circuit son et registre de données en lecture du circuit son.
$FF8802.B registre de données en écriture du circuit son.

Les mémoires $FF8800 à $FF88FF peuvent être lues ou écrites.

Périphérique multi-fonctions 68901

$FFFA01.B registre général d'entrée-sortie des données.
$FFFA03.B registre de sélection du front actif.

$FFFA05.B registre de direction des données.
$FFFA07.B registre de validation des interruptions (A).
$FFFA09.B registre de validation des interruptions (B).
$FFFA0B.B registre d'interruptions en attente de service (A).
$FFFA0D.B registre d'interruptions en attente de service (B).
$FFFA0F.B registre d'interruptions en cours de service (A).
$FFFA11.B registre d'interruptions en cours de service (B).
$FFFA13.B registre de masque des interruptions (A).
$FFFA15.B registre de masque des interruptions (B).
$FFFA17.B registre vecteur (bit 3 : mode, bits 4 à 7 : numéro).
$FFFA19.B registre de contrôle du timer A.
$FFFA1B.B registre de contrôle du timer B.
$FFFA1D.B registre de contrôle des timers C et D.
$FFFA1F.B registre de données du timer A.
$FFFA21.B registre de données du timer B.
$FFFA23.B registre de données du timer C.
$FFFA25.B registre de données du timer D.
$FFFA27.B registre de caractère de synchronisation.
$FFFA29.B registre de contrôle général de l'E/S série.
$FFFA2B.B registre d'état du récepteur série.
$FFFA2D.B registre d'état de l'émetteur série.
$FFFA2F.B registre de données de l'E/S série.

Circuit ACIA clavier :

$FFFC00.B registre de contrôle de l'ACIA clavier.
$FFFC02.B registre de données de l'ACIA clavier.

Circuit ACIA MIDI :

$FFFC06.B registre de contrôle de l'ACIA MIDI.
$FFFC08.B registre de données de l'ACIA MIDI.

ANNEXE 4
TRAPS

Tout appel de Trap, qu'il soit fait en assembleur ou en C, nécessite un ou plusieurs paramètres d'appel, le paramètre minimum étant le numéro de la fonction Trap appelée. Ces paramètres sont placés dans la pile avant l'appel selon le schéma suivant :

numéro de la fonction	haut de la mémoire
premier paramètre	
deuxième paramètre	
etc.	

où chaque case correspond à un mot, soit 2 octets.

Un paramètre de retour entier est toujours renvoyé dans DO. Certaines fonctions renvoient un paramètre long (noté DO.L dans les tableaux suivants). Lorsque d'autres paramètres de retour sont nécessaires, ils sont retournés par le biais de pointeurs dont les adresses ont été fournies parmi les paramètres d'appel.

Pour appeler un Trap en C, deux méthodes sont utilisables :

1) <paramètre de retour> = <nom de fonction> (<1er paramètre>, <2e paramètre>, etc...)

exemple : a = Bconstat (2) exécute Trap 13, avec 1 pour numéro de fonction et 2 comme 1er et seul paramètre. a est le paramètre de retour (ici l'état des entrées clavier).

2) <paramètre de retour> = <nom du Trap> (<numéro de fonction>, <1er paramètre>, <2e paramètre>, etc...)

exemple : a = bios (1,2) exécute Trap 13, avec 1 pour numéro de fonction et 2 comme paramètre (identique à ci-dessus).

Selon le compilateur C utilisé, les noms de Traps et les noms de fonctions système peuvent être différents. Avec le compilateur Digital Research, les Traps ont les noms suivants :

TRAP 1 : gemdos
TRAP 13 : bios
TRAP 14 : xbios

Les noms de fonctions C correspondant à chacune des fonctions système sont fournis dans les tableaux qui suivent.

On notera que TRAP 2 n'a pas de nom en C, ce qui s'explique par la non implémentation des fonctions CP/M par le biais de Trap 2. Toutes les fonctions VDI et AES sont appelées par l'intermédiaire d'un Trap 2, mais leur appel se fait par le nom de fonction VDI ou AES (cf. Annexe 2). Il est envisageable, en assembleur d'utiliser Trap 2 pour un appel de fonction GEM (cf. 3.1.3. pour une explication détaillée de la procédure à mettre en œuvre). Deux autres fonctions sont disponibles par Trap 2 en assembleur, elles sont étudiées dans cette annexe.

Une extrême vigilance s'impose quant à la longueur (mot ou long mot) des paramètres d'appel ou de retour des fonctions système. En effet, la plupart des compilateurs ne retournent pas de code erreur si l'on utilise un paramètre long à la place d'un paramètre court et réciproquement. Bien souvent, le résultat à l'exécution est catastrophique ! Il est cependant possible d'utiliser en C une variable entière pour stocker un paramètre de retour long si les deux octets inférieurs du paramètre de retour suffisent (exemple : Cconin() qui renvoie une entrée clavier, dont on peut négliger le mot fort). Il est également possible de ne pas employer de paramètre de retour du tout si l'on décide de ne pas en tenir compte, surtout lorsque ce paramètre est un simple code d'erreur (exemple : Tsettime (<heure>) peut être appelé ainsi en C, si l'on est certain que l'heure fournie est correctement formatée).

Avec TRAP 1 et TRAP 2, seul le registtre D0 est modifié durant l'exécution du Trap (il contient le paramètre de retour). Tous les autres registres reviennent inchangés, sauf pour Trap 1 fonction $20 (passage d'un mode à un autre) dans lequel DO, D1 et A7 sont modifiés.

Avec TRAP 13 et TRAP 14, les registres D0 à D2 et A0 à A2 peuvent être modifiés durant l'exécution du Trap. Par contre les registres D3 à D7 et A3 à A7 sont inchangés.

TRAP 1

TRAP 1 fonction 0 : Pterm0() TERM

Termine le programme en cours (cf Trap 1, fonction $4C)
paramètres d'appel : aucun
paramètre de retour : toujours nul

TRAP 1 fonction 1 : Cconin () CONIN

Lit un caractère entré au clavier et l'affiche à l'écran
paramètres d'appel : aucun
paramètre de retour : (long) caractère entré au clavier

Remarque : Le caractère est retourné sous la forme d'un long mot ayant la structure suivante : 00 <code clavier> 00 <code ASCII> ou le code clavier dépend de la touche appuyée :

code	touche	code	touche	code	touche	code	touche
01	Esc	19	P	31	N	4E	+ (pave)
02	1	1A	^	32	,	50	fl. bas
03	2	1B	$	33	;	52	Insert
04	3	1C	Return	34	:	53	Delete
05	4	1D	Control	35	=	60	<
06	5	1E	Q	36	Shift (d)	61	Undo
07	6	1F	S	38	Alternate	62	Help
08	7	20	D	39	Espace	63	((pavé)
09	8	21	F	3A	Capslock	64	) (pavé)
0A	9	22	G	3B	F1	65	/ (pavé)
0B	0	23	H	3C	F2	66	* (pavé)
0C	)	24	J	3D	F3	67	7 (pavé)
0D	–	25	K	3E	F4	68	8 (pavé)
0E	Backspace	26	L	3F	F5	69	9 (pavé)
0F	Tab	27	M	40	F6	6A	4 (pavé)
10	A	28	%	41	F7	6B	5 (pavé)
11	Z	29	'	42	F8	6C	6 (pavé)
12	E	2A	Shift(g)	43	F9	6D	1 (pavé)
13	R	2B	#	44	F10	6E	2 (pavé)
14	T	2C	W	47	Home	6F	3 (pavé)
15	Y	2D	X	48	fl.haut	70	0 (pavé)
16	U	2E	C	4A	– (pavé)	71	. (pavé)
17	I	2F	V	4B	fl.gauche	72	Enter(pavé)
18	O	30	B	4D	fl.droite		

Le retour de la fonction Cconin (1) ne s'effectue que lorsqu'une touche du clavier a été pressée (sauf Shift, Control, Alternate et Capslock, qui ne provoquent pas un retour de fonction).

TRAP 1 fonction 2 : Cconout (a) CONOUT

Affiche un caractère à l'écran

paramètres d'appel : (mot) caractère à afficher
paramètre de retour : aucun

L'octet faible du paramètre d'appel correspond au code ASCII du caractère à afficher, l'octet fort doit être nul.

TRAP 1 fonction 3 : Cauxin () AUXIN

Lit un caractère sur l'entrée série RS 232

paramètres d'appel : aucun
paramètre de retour : (long) caractère lu

Le retour de cette fonction ne s'effectue que lorsqu'un caractère a été reçu sur l'entrée série. Ce caractère est retourné dans l'octet faible de D0.

TRAP 1 fonction 4 : Cauxout (a) AUXOUT

Écrit un caractère sur la sortie série RS 232

paramètres d'appel : (mot) caractère à envoyer
paramètre de retour : aucun

Le caractère est placé dans l'octet faible. L'octet fort doit être nul.

TRAP 1 fonction 5 : Cprnout (a) PRTOUT

Écrit un caractère sur la sortie parallèle.

paramètres d'appel : (mot) caractère à envoyer
paramètre de retour : (long) −1 si envoi correct, 0 sinon

Le caractère est placé dans l'octet faible. L'octet fort doit être nul. Si l'imprimante n'est pas connectée ou si la ligne Busy reste active durant 30 secondes, un retour de fonction s'effectue avec D0.L = 0.

TRAP 1 fonction 6 : Crawio (a) RAWCONIO

Lecture d'une entrée clavier brute ou affichage

paramètres d'appel : (mot) type d'entrée-sortie demandée
si (−1) lit une entrée clavier
sinon affiche le paramètre d'appel à l'écran
paramètre de retour : (mot) entrée clavier si par. d'appel = −1

Si le paramètre d'appel est égal à −1 (demande d'entrée clavier), le retour de fonction s'effectue avec le caractère lu dans le tampon clavier ou 0 si aucun caractère n'est dans ce tampon. Le retour est donc immédiat et le caractère n'est pas affiché à l'écran (pas d'écho).

TRAP 1 fonction 7 : Crawcin () CONIN_WE

Lecture d'une entrée clavier brute (sans écho)

paramètres d'appel : aucun
paramètre de retour : (long) caractère lu au clavier

Le caractère lu au clavier est retourné selon la même structure que pour la fonction Cconin (page précédente). Le caractère n'est pas affiché à l'écran (pas d'écho). Le retour de la fonction ne s'effectue qu'après une entrée clavier.

TRAP1 fonction 8 : Cnecin ()

Lecture d'une entrée clavier sans écho.

paramètres d'appel : aucun
paramètre de retour : (long) caractère lu au clavier

Le caractère lu au clavier est retourné selon la même structure que pour la fonction Cconin et non affiché. Les codes de contrôle, ^C, ^Q et ^S sont interprêtés (sortie du programme, reprise et arrêt d'affichage) et ne provoquent pas un retour de la fonction.

TRAP 1 fonction 9 : Cconws (a) PRINTLINE

Affichage d'une chaine à l'écran

paramètres d'appel : (long) adresse de la chaine à afficher
paramètre de retour : aucun

La chaîne à afficher est envoyée caractère par caractère. Le dernier caractère de la chaîne doit être suivi d'un caractère nul (00). Chaque caractère de la chaîne doit être codé sur un octet.

TRAP 1 fonction $A : Cconrs (a) READLINE

Lecture d'une chaîne entrée au clavier

paramètres d'appel : (long) adresse du tampon où stocker la chaîne
paramètre de retour : aucun

En entrée, le premier octet du tampon doit contenir le nombre maximum d'octets à lire ou, par défaut, le nombre d'octets du tampon moins deux. En sortie, le nombre d'octets lu est indiqué dans le deuxième octet du tampon. A partir du troisième octet est stockée la chaîne.

Le retour de la fonction s'effectue lorsqu'un retour de chariot est entré au clavier. Si le nombre maximum d'octets à lire est atteint avant un retour de chariot, une sonnerie (^G) résonne et les octets suivants ne sont pas pris en compte. Le retour de fonction s'opère au premier retour de chariot.

Les codes de contrôle^G, ^E, ^Q, ^R, ^S, ^U, ^X, le retour arrière et l'effacement ne provoquent pas d'entrées dans la chaîne mais sont interprétés selon leur fonction normale.

TRAP 1 fonction $B : Cconis () CONSTAT

Demande l'état du tampon clavier

paramètres d'appel : aucun
paramètre de retour : (mot) −1 si un caractère est disponible, 0 sinon

TRAP 1 fonction $E : Dsetdrv (a) SETDRV

Fixe le lecteur de disque par défaut

paramètres d'appel : (mot) numéro du disque (0 si A, 1 si B, etc.)
paramètre de retour : (long) carte des lecteurs actifs

La carte des lecteurs actifs renvoyée dans D0 est la suivante : le bit 0 est à 1 si le lecteur A est actif, le bit 1 est à 1 si le lecteur B est actif, etc... Les lecteurs C, D, ..., P suivants ne sauraient naturellement correspondre qu'à des disques virtuels (cf. chapitre 3).

TRAP 1 fonction $10 : Cconos () CONOSTAT

Demande l'état de l'écran

paramètres d'appel : aucun
paramètre de retour : (long) −1 si l'écran est prêt à recevoir un caractère, 0 sinon

Type même de la fonction inutile ! Retourne toujours −1 ...

TRAP 1 fonction $11 : Cprnos ( ) PRTOSTAT

Demande l'état de la ligne Busy de la sortie parallèle

paramètres d'appel : aucun
paramètre de retour : (long) −1 si imprimante prête, 0 sinon

Type même de la fonction peu utile ! Une vérification de l'état de la ligne Busy est en effet effectuée dans tous les cas lors de l'écriture sur la sortie parallèle (cf. Cprnout, page précédente).

TRAP 1 fonction $12 : Cauxis ( ) AUXISTAT

Demande l'état du tampon d'entrée série RS 232

paramètres d'appel : aucun
paramètre de retour : −1 si un caractère est disponible, 0 sinon.

Comme pour la fonction précédente, toute demande d'entrée série débute par un test d'état du tampon, répété jusqu'à ce qu'un caractère soit disponible. La fonction Cauxis a toutefois l'intérêt de permettre de vérifier que l'entrée sortie série est bien reliée à un périphérique (avec une boucle de temporisation adéquate).

TRAP1 fonction $13 : Cauxos ( ) AUXOSTAT

Demande l'état du tampon de sortie série RS 232

paramètres d'appel : aucun
paramètre de retour : −1 si la sortie série est prête à recevoir 0 sinon

Retourne −1 si le tampon de sortie série n'est pas plein.

TRAP 1 fonction $19 : Dgetdrv (a) CURDRV

Recherche le lecteur de disque par défaut

paramètres d'appel : aucun
paramètre de retour : (mot) 0 si disque A, 1 si disque B, etc.

cf. la fonction corollaire Dsetdrv( ), Trap 1 fonction $E.

TRAP 1 fonction $1A : Fsetdta (a) SETDTA

Fixe l'adresse de stockage des données sur un fichier

paramètres d'appel : (long) adresse du tampon de stockage
paramètre de retour : aucun

Le tampon de stockage des données d'un fichier de disque est utilisé par les fonctions Fsfirst et Fsnext (Trap 1 fonctions $4E et $4F). Ce tampon doit comporter au minimum 44 octets.

Un tampon de stockage par défaut est créé à l'initialisation du système. Son adresse peut être obtenue par Fgetdta (Trap 1, fonction $2F).

TRAP 1 fonction $20 : Super (a) SUPER

Changement de mode (superviseur en utilisateur, ou le contraire)

Les paramètres d'appel et de retour de cette fonction diffèrent selon le mode de fonctionnement du microprocesseur 68000.

Si l'on se trouve en *mode utilisateur* :
paramètres d'appel : (long) pointeur de pile SUPERVISEUR
ou 0 si l'on désire user du pointeur de pile superviseur système.
paramètre de retour : ancien pointeur de pile utilisateur.

Si l'on se trouve en *mode superviseur* :

paramètres d'appel : (long) pointeur de pile UTILISATEUR qui doit être le paramètre de retour précédent si l'on est passé en superviseur par Super(a) avec un changement de pointeur de pile superviseur.
paramètre de retour : aucun

Si la fonction Super est employée avec un paramètre d'appel égal à 1, il s'agit alors d'un *test de mode.* Le paramètre de retour (long) est égal à 0 si l'on se trouve en mode UTILISATEUR et −1 si l'on se trouve en mode SUPERVISEUR.

TRAP 1 fonction $2A : Tgetdate( ) GET_DATE

Demande la date

paramètres d'appel : aucun
paramètre de retour : (mot) date

La date est ainsi formatée dans un mot :

bits 0 à 4 : jour du mois (1 à 31)
bits 5 à 8 : mois de l'année (1 à 12)
bits 9 à 15 : année − 1980 (0 à 119 avec 1980=0 et 2099=119)

La date fournie n'a naturellement de sens que si la date correcte a été auparavant initialisée par Tsetdate ci-dessous.

TRAP 1 fonction $2B : Tsetdate (a) SET-DATE

Fixe la date

paramètres d'appel : (mot) date selon format ci-dessus
paramètre de retour : −1 si le format de la date fournie est incorrect
0 si le format de la date fournie est correct

TRAP 1 fonction $2C : Tgettime( ) GET_TIME

Demande l'heure

paramètres d'appel : aucun
paramètre de retour : heure selon le format suivant :
bits 0 à 4 : nombre de secondes (arrondi pair)
bits 5 à 10 : nombre de minutes
bits 11 à 15 : nombre d'heures

TRAP 1 fonction $2D : Tsettime (a) SET_TIME

Fixe l'heure

paramètres d'appel : (mot) heure selon format ci-dessus
paramètre de retour :
−1 si le format de l'heure fournie est invalide
0 si le format de l'heure fournie est valide

TRAP 1 fonction $2F : Fgetdta ( ) GETDTA

Demande l'adresse du tampon de stockage des données d'un fichier

paramètres d'appel : aucun
paramètre de retour : (long) adresse du tampon

cf. Fsetdta, Trap 1 fonction $1A.

TRAP 1 fonction $30 : Sversion ( ) GETVNR

Demande le numéro de version du GEMDOS

paramètres d'appel : aucun
paramètre de retour : (mot) numéro de version

L'octet faible de D0 contient le numéro maître et l'octet faible le numéro secondaire (exemple : 0001 correspond à la version 1.00).

Des six versions du système d'exploitation sur lesquelles a travaillé l'auteur de Juillet 85 à Janvier 86, deux portaient le numéro de version 0D00 soit 0.13 et quatre le numéro 1300 soit en principe 0.19. Il est toutefois vraisemblable qu'il s'agit de la même version dont le numéro a été "décimalisé".

TRAP 1 fonction $31 : Ptermres (a,b) KEEPPROC

Fin d'un programme et libération de mémoire

paramètres d'appel : (long) nombre d'octets de mémoire à libérer
(mot) code de sortie du programme
paramètre de retour : aucun (naturellement)

Fonction dangereuse puisque son résultat n'est pas garanti dans les systèmes d'exploitation futurs ! La libération de mémoire est étudiée dans Mshrink (Trap 1, fonction $44). Utiliser Pterm0 ou Pterm(a) (Trap 1, fonctions 0 et $4C) de préférence.

TRAP 1 fonction $36 : Dfree (a,b) GETDFREE

Demande d'informations sur un disque

paramètres d'appel : (long) adresse du tampon ou stocker les info.
(mot) numéro du disque (0 par défaut, 1 si A, 2 si B, etc.)
paramètre de retour : −46.L si le disque n'est pas actif, 0 sinon

Le tampon dont l'adresse est envoyée en appel doit comporter au moins seize octets. Si l'adresse de départ de ce tampon est notée "ad", on aura :

ad	→ nombre de blocs libres sur le disque
ad + 4	→ nombre total de blocs du disque
ad + 8	→ taille d'un secteur en octets
ad + 12	→ nombre de secteurs par bloc

Un secteur contient 512 octets ; un bloc est formé de 2 secteurs, soit 1024 octets. Le bloc est l'unité de chargement d'un fichier sur disque, c'est-à-dire qu'un fichier même de 10 octets occupera un minimum de 1024 octets sur le disque. Au formatage, 351 blocs existent sur un disque simple face et 711 blocs sur un disque double face (nombre retourné dans ad + 4). Mais l'en-tête de disque occupe dans les deux cas 2 blocs. On dispose donc de 349 blocs réels (349 Ko) sur un disque simple face et de 709 blocs (709 Ko) sur un disque double face. Les titres et informations sur les programmes sont stockés dans l'en-tête du disque et n'appartiennent pas aux blocs. Ainsi un fichier de 1024 octets n'occupera qu'un bloc... tout comme un fichier de 2 octets !

TRAP 1 fonction $39 : Dcreate (a) MKDIR

Création d'un sous-répertoire de disque

paramètres d'appel : (long) adresse de la chaine contenant le nom
paramètre de retour : (long) 0 si création bien effectuée
−34L si nom de disque non trouvé
−36L si disque non actif

Le nom du sous-répertoire doit débuter par la lettre identifiant le disque (A, B...), en majuscule ou minuscule, suivi de ":" et du nom du sous-répertoire, sauf si le disque concerné est le disque courant (cf. Dsetdrv et Dgetdrv), auquel les deux premiers caractères ne sont pas nécessaires. La chaîne doit se terminer par un caractère nul.

TRAP 1 fonction $3A : Ddelete (a) RMDIR

Suppression d'un sous-répertoire de disque

paramètres d'appel : (long) adresse de la chaine contenant le nom
paramètre de retour : (long) 0 si effacement correct, négatif sinon

Supprime un sous-répertoire créé par la fonction ci-dessus.

TRAP 1 fonction $3B : Dsetpath (a) CHDIR

Fixe le répertoire courant d'un disque

paramètres d'appel : (long) adresse de la chaîne contenant le nom
paramètre de retour : (long) 0 si création correcte, négatif sinon

Le nom du répertoire doit être contenu dans une chaîne terminée par un caractère nul. Les notes afférentes à Dcreate (Trap 1, fonction $39) sont également applicables à Dsetpath.

TRAP 1 fonction $3C : Fcreate (a,b) CREATE

Création d'un fichier sur disque

paramètres d'appel : (long) adresse de la chaîne contenant le nom
(mot) type du fichier créé :
00 fichier normal
01 fichier ne pouvant qu'être lu
02 fichier transparent au répertoire
04 fichier système, transparent au répertoire
08 fichier contenant un nom dans les 11 premiers octets
paramètre de retour : (long) négatif si création non effectuée
identificateur du fichier si création faite

La chaîne contenant le nom du fichier doit commencer par la lettre identifiant le disque (A, B...) et le symbole ":" puis le nom de répertoire et le nom de fichier, un caractère nul terminant la chaîne. La non création d'un fichier peut résulter du manque d'espace dans le répertoire, de l'impossibilité d'accéder au disque ou de l'existence du fichier à créer.

Un fichier non existant doit toujours être créé mais cette création fait office d'ouverture.

TRAP 1 fonction $3D : Fopen (a,b) OPEN

Ouverture d'un fichier existant sur disque

paramètres d'appel : (long) adresse de la chaîne contenant le nom (mot) type d'ouverture :
0 : ouverture en lecture seule,
1 : ouvert en écriture seule,
2 : ouvert en lecture/écriture
paramètre de retour : (long) identificateur du fichier ouvert négatif si fichier non ouvert.

Il est plus que recommandé de conserver le paramètre de retour puisqu'il s'agit de l'identificateur du fichier, indispensable pour toute opération ultérieure sur le fichier. L'identificateur de fichier correspond au mot faible du

paramètre de retour. Après ouverture, le pointeur de fichier est positionné en début de fichier (cf Fseek).

TRAP 1 fonction $3E : Fclose (a) CLOSE

Fermeture d'un fichier sur disque

paramètres d'appel : (mot) identificateur du fichier
paramètre de retour : (long) 0 si fermeture correcte
négatif sinon

L'identificateur de fichier envoyé en appel correspond à l'identificateur renvoyé à l'ouverture du fichier.

TRAP 1 fonction $3F : Fread (a,b,c) READ

Lecture d'octets dans un fichier sur disque

paramètres d'appel : (mot) identificateur du fichier (cf. Fopen)
(long) nombre d'octets à lire
(long) adresse du tampon de stockage
paramètre de retour : (long) nombre d'octets lus négatif si erreur.

Les octets sont lus dans le fichier à partir du pointeur de fichier (cf. Fseek, Trap 1, fonction $42). Ils sont stockés à partir de l'adresse fournie dans le 3e paramètre d'appel.

TRAP 1 fonction $40 : Fwrite (a,b,c) WRITE

Écriture d'octets dans un fichier sur disque

paramètres d'appel : (mot) identificateur du fichier (cf. Fopen)
(long) nombre d'octets à écrire
(long) adresse du tampon de stockage
paramètre de retour : (long) nombre d'octets écrits négatif si erreur.

Les octets sont placés dans le fichier à partir de la position du pointeur de fichier (cf. Fseek, fonction $42). Ils sont lus dans le tampon dont l'adresse est fournie en 3e paramètre d'appel.

TRAP 1 fonction $41 : Fdelete (a) UNLINK

Effacement d'un fichier sur disque

paramètres d'appel : (long) adresse de la chaîne contenant le nom
paramètre de retour : (long) 0 si le fichier est effacé négatif si erreur (fichier inexistant...)

La chaîne contenant le nom du fichier doit commencer par la lettre d'identification du disque (A,B,...) et le symbole par défaut (cf. Dsetdrv). Elle doit se terminer par un octet nul.

TRAP 1 fonction $42 : Fseek (a,b,c) LSEEK

paramètres d'appel : (long) nombre d'octets
(mot) identificateur du fichier (cf. Fopen)
(mot) type de pointeur de fichier
paramètre de retour : position du pointeur de fichier par rapport au début de celui-ci, négatif si erreur.

Le type de pointeur de fichier (3e paramètre d'appel) définit le type de positionnement désiré :
0 : place le pointeur N octets après le début de fichier
1 : place le pointeur N octets après la position actuelle du pointeur
2 : place le pointeur N octets avant la fin de fichier (N négatif)
où N est le nombre d'octets entré comme 1er paramètre d'appel. En utilisant Fseek (0, <id. fichier>, 2), on obtiendra en retour le nombre d'octets du fichier.

TRAP 1 fonction $43 : Fattrib (a,b,c) CHMOD

Fixe ou demande les attributs d'un fichier

paramètres d'appel : (long) adresse de la chaîne contenant le nom
(mot) 0 pour demander les attributs
1 pour fixer les attributs
(mot) attributs à fixer (0 si demande)
paramètre de retour : (long) attributs si demande

Les attributs sont définis ainsi :
01 : fichier à lecture seule
02 : fichier transparent au répertoire
04 : fichier système, transparent au répertoire
08 : début de fichier correspondant au nom (11 octets)
10 : début de fichier correspondant à un sous-répertoire
20 : fichier écrit puis refermé

TRAP 1 fonction $45 : Fdup (a) DUP

Génère une copie d'un identificateur de fichier

paramètres d'appel : (mot) identificateur du fichier (cf. Fopen)
paramètre de retour : (long) nouvel identificateur du fichier
négatif si erreur

A un fichier déjà ouvert est affecté un second identificateur, le pointeur de fichier affecté au 2e identificateur suivant les modifications affectant le pointeur affecté au 1er identificateur. Un identificateur de fichier standard est toujours compris entre 0 et 5, alors que les doublons sont supérieurs ou égaux à 6. L'utilité du gemellage ainsi effectué est restreinte...

TRAP 1 fonction $46 : Fforce (a,b) FORCE

Force un identificateur standard à pointer le même fichier qu'un identificateur de fichier non standard

paramètres d'appel : (mot) identificateur de fichier standard
(mot) identif. de fichier non standard
paramètre de retour : (long) 0 si opération bien effectuée
négatif sinon.

TRAP 1 fonction $47 : Dgetpath (a,b) GETDIR

Demande du répertoire courant

paramètres d'appel : (long) adresse du tampon de stockage
(mot) 0 si disque par défaut,
1 si disque A, 2 si disque B...
paramètre de retour : négatif si disque non trouvé, 0 sinon.

Le tampon de stockage doit être d'au moins 64 octets. La chaîne retournée ne comprend pas la lettre d'identification du disque (A, B...) ni les deux points et la barre inversée (\) de début de répertoire.

TRAP 1 fonction $48 : Malloc (a) MALLOC

Demande d'allocation mémoire

paramètres d'appel : (long) nombre d'octets à allouer
−1 pour savoir le nombre d'octets disponibles
paramètre de retour : (long) adresse de départ zone mémoire allouée ou nombre d'octets disponibles (si −1 à l'appel)
0 si allocation mémoire impossible

Une réservation mémoire s'opère habituellement en deux phases :

1) demande du nombre d'octets disponibles (Malloc (−1)),
2) réservation par : adresse = Malloc (<nombre d'octets>).

Il est déconseillé de réserver toute la mémoire disponible, certaines fonctions GEM (comme fsel_input) ayant besoin d'effectuer des allocations mémoire temporaires (laisser au moins 3 Ko en réserve). Le nombre d'octets alloués est toujours pair (arrondi supérieur).

TRAP 1 fonction $49 : Mfree (a) MFREE

Libération de mémoire allouée précédemment

paramètres d'appel : (long) adresse de départ de la zone allouée
paramètre de retour : (long) 0 si mémoire libérée, négatif si erreur

Cette fonction doit être utilisée avant sortie d'un programme ou une zone mémoire a été allouée par Malloc (cf. ci-dessus).

TRAP 1 fonction $4A : Mshrink (a,b) SETBLOCK

Libération de mémoire au profit du système

paramètres d'appel : (mot) 0 (réservé pour le système)
(long) adresse de base de la zone à libérer
(long) nombre d'octets à libérer
paramètre de retour : 0 si libération effectuée,
négatif sinon

Cette fonction sert à réaffecter une zone mémoire du programme du système. En fait, le prologue de tout programme compilé et "lié" contient cette fonction (ajoutée par le compilateur). Tout programme démarre donc avec la taille mémoire juste nécessaire au code objet, aux données et à la pile. L'intérêt de cette fonction réside dans le fait qu'elle permet de rendre au système une part d'une zone mémoire préalablement réservée par Malloc.

TRAP 1 fonction $4B : Pexec (a,b,c,d) EXEC

Charge et exécute un programme

paramètres d'appel : (mot) 0 pour chargement et exécution
3 pour chargement sans exécution
(long) adresse de la chaîne contenant le nom
(long) adresse d'un jeu de commandes
(long) adresse des chaînes de configuration
paramètre de retour : (long) adresse page de base si chargement code du programme appelé si chargement et exécution, négatif si erreur

Les principes de chargement et d'exécution d'un programme ainsi que la signification des pointeurs ci-dessus sont détaillés au chapitre 4.

TRAP 1 fonction $4C : Pterm (a) TERM-2

Termine un programme

paramètres d'appel : (mot) code du programme retourné par Pexec
paramètre de retour : négatif si erreur, 0 sinon.

cf. fonction Pexec ci-dessus.

TRAP 1 fonction $4E : Fsfirst (a,b) SFIRST

Recherche la première occurrence d'un fichier

paramètres d'appel : (long) adresse de la chaîne contenant le nom
(mot) attributs de la recherche
paramètre de retour : 0 si fichier trouvé, négatif sinon.

Les attributs de recherche correspondent aux attributs de fichier (voir fonction $43, Fattrib). La recherche d'une première occurrence n'a vraiment de sens que lorsque l'on emploie des " * " ou des "?" dans le nom de fichier. Ce nom doit commencer par la lettre identifiant le disque (A, B...) puis ":", etc. comme pour tout nom de fichier.

Si un premier fichier est trouvé, un bloc d'informations de 64 octets est placé à partir de l'adresse fixée par Fsetdta (fonction $1A) ou fournie par Fgetdta (fonction $2F). Ce bloc a la structure suivante :

octets 0 à 20 : inutilisés (réservés)
octet 21 : attributs du fichier (cf. Fattrib, fonction $43)
octets 22 à 23 : heure de création du fichier
octets 24 à 25 : date de création du fichier
octets 26 à 29 : taille du fichier, en octets
octets 30 à 43 : nom et extension du fichier trouvé

TRAP 1 fonction $4F : Fsnext ( ) SNEXT

Recherche l'occurrence suivante d'un fichier

paramètres d'appel : aucun
paramètre de retour : (long) 0 si fichier trouvé, négatif sinon.

La recherche est effectuée d'après les paramètres d'appel de Fsfirts (voir ci-dessus) qui a dû obligatoirement précéder l'appel de Fsnext. Les informations en retour sont également stockées à partir de l'adresse placée par Fsetdta ou fournie par Fgetdta.

TRAP 1 fonction $56 : Frename (a,b,c) RENAME

Renomme un fichier

paramètres d'appel : (mot) 0, réservé pour le système
(long) adresse du nom du fichier existant
(long) adresse du nouveau nom du fichier

paramètre de retour : (long) 0 si opération correcte, négatif sinon.

L'identificateur du disque (A,B...) doit être identique pour les deux noms de fichiers. Attention à ne pas utiliser un nom de fichier déjà existant lorsque vous renommez un fichier car il serait recouvert par le fichier renommé...

TRAP 1 fonction $57 : Fdatime (a,b,c) GSDTOF

Fixe ou demande la date et l'heure de création d'un fichier

paramètres d'appel : (long) adresse du tampon de stockage date/heure
(mot) identificateur du fichier
(mot) 1 pour fixer la date et l'heure.
0 pour demander la date et l'heure.

L'heure est formatée dans le premier mot du tampon et la date dans le deuxième mot, selon le format décrit dans les fonctions $2A et $2C.

TRAP 2

TRAP 2 fonction $0 GETAPB

Termine un programme

Fonction strictement équivalent à Pterm (Trap 1, fonction $4C), avec un code d'appel égal à 0. Termine le programme en cours et rend le contrôle au système.

TRAP 2 fonction $73

paramètre d'appel : DO.W = $0073
paramètre de retour : retourné dans intout(0)

Exécution d'une fonction GEM VDI. Les paramètres d'appel et de retour sont placés dans les variables de base (contrl, intin, ptsin, intout, ptsout). L'adresse d'une table pointant les adresses de contrl, intin, etc., doit être placée dans D1.L (cf. 3.1.3). Se reporter à l'annexe 2 pour le détail de ces paramètres d'appel et de retour.

TRAP 2 fonction $C8

paramètre d'appel : DO.W = $00C8
paramètre de retour : retourné dans intout (0)

Exécution d'une fonction GEM AES. Les paramètres d'appel et de retour sont placés dans les variables de base de l'AES (global, control, params, int in, int_out, adr_in, adr_out). L'adresse d'une table de pointeurs des adresses de global, control, etc., (voir 3.1.3) doit être placée dans D1.L Se reporter à l'annexe 2 pour le détail de ces paramètres d'appel et de retour.

TRAP 2 fonction $C9

paramètre d'appel : DO.W = $00C9
paramètre de retour : retourné dans intout (0)

Réservée au système

TRAP 2 fonction $FFFF

Fournit l'adresse de départ des routines VDI.

paramètre d'appel : DO.W = $FFFF
paramètre de retour : DO.L = adresses du "dispatcher" des routines VDI.

Attention, contrairement, à *toutes* les autres fonctions des Traps, cette fonction réclame un paramètre d'appel ayant l'octet fort *signé* (en principe, l'octet fort est toujours nul). L'adresse longue fournie en retour est celle de la routine système de traitement des fonctions VDI. Dans ce cas, pour une exécution légèrement accélérée des fonctions VDI. Dans ce cas, *il faut être en mode superviseur* (cf. Trap 1, fonction $20, Super ( )).

TRAP 13

TRAP 13 fonction $0 GETMPB

Informations sur la mémoire

paramètres d'appel : (long) adresse d'un tampon de stockage
paramètre de retour : (long) nombre d'octets disponibles en mémoire

Le tampon de 12 octets minimum, est rempli ainsi :

(long) adresse du tampon de données mémoire MD décrit ci-dessous,
(long) 0,
(long) adresse du tampon de données mémoire MD décrit ci-dessous.

Le tampon de données mémoire MD est structuré ainsi :

(long) 0,
(long) adresse de départ de la mémoire libre
(long) nombre d'octets de mémoire libre
(long) 0.

TRAP 13 fonction $1 : Bconstat (a) BCONSTAT

Demande l'état d'un périphérique en entrée

paramètres d'appel : (mot) code du périphérique
paramètre de retour : (long) 0 si périphérique prêt (ou caractère disponible, −1 sinon.

Le code du périphérique (paramètre d'appel) peut être :

0	PRN :	sortie parallèle
1	AUX :	entrée-sortie série RS 232C
2	CON :	entrée-sortie console (ici clavier)
3		MIDI (entrée-sortie interface musicale)
4		port clavier (envoi de commande au 6301)
5		sortie écran de tous caractères

Les différents périphériques sont validés ou non suivant la fonction appelée (Bconstat, Bconin, Bconout). On a :

		PRN	AUX	CON	MIDI	6301	écran
test entrée	:	non	oui	oui	oui	non	non
entrée	:	oui	oui	oui	oui	non	non
test sortie	:	oui	oui	oui	oui	oui	non
sortie	:	oui	oui	oui	oui	oui	oui

TRAP 13 fonction $2 : Bconin (a) BCONIN

Attente d'entrée d'un caractère sur un périphérique

paramètres d'appel : (mot) code du périphérique (voir ci-dessus)
paramètre de retour : (long) code du caractère (voir Trap. 1, fonction 1)

L'attente du caractère est infinie, y compris pour l'imprimante.

TRAP 13 fonction $3 : Bconout (a,b) BCONOUT

Envoi d'un caractère sur un périphérique

paramètres d'appel : (mot) code du périphérique (voir Trap 13, $1)
(mot) caractère à envoyer

Pour tous les périphériques, le retour de la fonction ne se fait qu'une fois le caractère envoyé, excepté pour l'imprimante ou le retour s'opère au bout de 30 secondes si la ligne Busy reste à l'état haut. La différence entre le périphérique CON (2) en sortie et le périphérique "écran" (5) tient à ce qu'avec ce dernier, tous les codes envoyés sont affichés, même les codes de contrôle. Il n'y a donc plus alors d'éditeur (cf. chapitre 5).

TRAP 13 fonction 4 : Rwabs (a,b,c,d,e) RWABS

Lecture ou écriture de secteur(s) sur un disque

paramètres d'appel : (mot) type d'opération (0 : lecture, 1 : écriture),
(long) adresse du tampon de stockage,
(mot) nombre de secteurs à lire ou écrire,
(mot) numéro du secteur de départ,
(mot) code du périphérique (0 = disque A, 1 = B,...)

paramètre de retour : 0 si l'opération correcte, négatif sinon.

TRAP 13 fonction 5 : Setexc (a,b) SETEXEC

Fixe ou lit un vecteur

paramètres d'appel : (mot) numéro du vecteur,
(long) adresse à placer (−1 si lecture)

paramètre de retour : (long) adresse du vecteur si lecture
ancienne adresse si écriture

Les vecteurs $00 à $FF sont les vecteurs 68000. Les vecteurs $100 à $1FF sont utilisés par le GEMDOS. Les principaux vecteurs sont décrits au chapitre 5.

TRAP 13 fonction 6 TICKCAL

Fournit la base de temps du timer en millisecondes

paramètres d'appel : aucun

paramètre de retour : (long) nombre de millisecondes de la base de temps du timer (normalement $14, puisque 50 Hz donnent une période de 20 ms).

TRAP 13 fonction 7 GET BPB

Fournit l'adresse du bloc de paramètres d'un disque

paramètres d'appel : (mot) numéro du disque (A=0, B=1, etc.)
paramètre de retour : (long) adresse du bloc de paramètres du disque 0 si le bloc n'existe pas.

La structure du bloc de paramètres d'un disque est :

0	(mot)	nombre d'octets par secteur
2	(mot)	nombre de secteurs par bloc (cf. TRAP 1, $36)
4	(mot)	nombre d'octets par bloc
6	(mot)	nombre de secteurs réservés au catalogue
8	(mot)	nombre de secteurs réservés aux tables d'allocation fichiers
A	(mot)	identique au précédent + 1 (boot secteur)
C	(mot)	nombre total de secteurs réservés du disque
E	(mot)	nombre total de bloc du disque (non réservés)
10	(mot)	0
12	(mot)	nombre de secteurs par piste
14	(mot)	nombre de faces du disque
16	(mot)	nombre de secteurs par piste (9 si simple face, 18 si double face)
18	(mot)	nombre de secteurs par piste
1A	(mot)	nombre de secteurs "cachés" (non pris en compte)

TRAP 13 fonction $8 : Bcostat (a) BCOSTAT

Demande l'état d'un périphérique en sortie

paramètres d'appel : (mot) code du périphérique (cf. TRAP 13, 1),
paramètre de retour : (long) −1 si périphérique prêt à recevoir
0 sinon.

TRAP 13 fonction $9 : Mediach (a) MEDIACH

Teste s'il y a eu changement de disque dans un lecteur

paramètres d'appel : (mot) 0 si lecteur A testé, 1 si lecteur B,
paramètre de retour : (long) 0 si disque non changé, 1 si le disque peut avoir changé, 2 si le disque a changé.

Le test permet de déterminer si le disque a été changé dans un lecteur donné depuis la dernière opération sur ce lecteur.

TRAP 13 fonction $A : Drvmap( ) DRVMAP

Retourne la carte descriptive des disques actifs

paramètres d'appel : aucun

paramètre de retour : (long) vecteur descriptif (bit 0 à 1 si le disque A est actif, bit 1 à 1 si le disque B est actif, etc.).

TRAP 13 fonction $B : Getshift (a) KB SHIFT

Active une des touches spéciales du clavier

paramètres d'appel : (mot) mode de fonctionnement du clavier
paramètre de retour : (long) ancien mode de fonctionnement.

Si le paramètre d'appel est négatif, le mode de fonctionnement est lu. Sinon, il est fixé en fonction de chacun des bits du paramètre :

bit 0 : touche Shift droite (appuyée si 1)
bit 1 : touche Shift gauche
bit 2 : touche Control
bit 3 : touche Alternate
bit 4 : touche Capslock
bit 5 : touche Clr/Home
bit 6 : touche Insert

Remarque : Le compilateur C de Digital Research « squeeze » le paramètre d'appel de la fonction, qui ne peut donc plus être appelée, sauf modification du fichier « osbinds.h ».

TRAP 14

TRAP 14 fonction $0 : Initmous (a,b,c)

Initialisation du gestionnaire de souris

paramètres d'appel : (mot) type d'initialisation :
0 désactive la souris
1 active la souris, en mode relatif
2 active la souris, en mode absolu
4 active la souris, en mode clavier
(long) adresse d'un bloc de paramètres
(long) 0.
paramètre de retour : 0 si initialisation correcte, −1 sinon.

Le bloc de paramètres pointé par le 2e paramètre d'appel à la structure suivante :

00 (octet) 0 si origine verticale de la position de la souris en bas de l'écran, 1 si origine verticale en haut de l'écran,
01 (octet) précise le mode d'action des boutons de la souris sur son déplacement (cf. commande 7 envoyée au 6301, chapitre 5).
02 (octet) déplacement horizontal de base de la souris
03 (octet) déplacement vertical de base de la souris

Si le type d'initialisation est égal à 2, les paramètres suivants doivent figurer à la suite du bloc précédent :

04 (mot) position horizontale maximale de la souris
06 (mot) position verticale maximale de la souris
08 (mot) position horizontale de départ de la souris
0A (mot) position verticale de départ de la souris

TRAP 14 fonction $1 SSBRK

Réservation de mémoire en deçà du haut de la mémoire vive

paramètres d'appel : (mot) nombre d'octets à réserver
paramètre de retour : (long) adresse du bas de la zone allouée

Fonction non implémentée dans les systèmes d'exploitation du ST.

TRAP 14 fonction $2 : Physbase ( )

Fournit la base physique de la mémoire d'écran.

paramètres d'appel : aucun
paramètre de retour : (long) adresse du bas de la mémoire écran.

TRAP 14 fonction $3 : Logbase ( )

Fournit la base logique de la mémoire d'écran

paramètres d'appel : aucun
paramètre de retour : (long) base logique de l'écran

Base physique et logique de l'écran coïncident sur le 520 ST.

TRAP 14 fonction $4 : Getrez ( )

Fournit la résolution courante de l'écran

paramètres d'appel : aucun

paramètre de retour : (mot) 2 si haute résolution, 1 si moyenne résolution, 0 si basse résolution.

TRAP 14 fonction $5 : Setscreen (a,b,c)

Fixe les données de base de l'écran

paramètres d'appel : (long) base logique de l'écran
(long) base physique de l'écran
(mot) résolution de l'écran (cf. ci-dessus)

paramètre de retour : aucun

Si l'un des paramètres est égal à −1, il n'est pas pris en compte.

TRAP 14 fonction $6 : Setpallete (a)

Fixe les couleurs de la palette courante

paramètres d'appel : (long) adresse de la palette
paramètre de retour : aucun

La palette adressée par le paramètre d'appel doit être composée de 16 mots, chacun des mots définissant une couleur (cf. 1.1.2. Système graphique).

TRAP 14 fonction $7 : Setcolor (a,b)

Fixe une couleur de la palette courante

paramètres d'appel : (mot) numéro de la couleur (0 à 15)
(mot) nouvelle couleur

paramètre de retour : (mot) données de la couleur modifiée.

Une couleur est définie par 3 groupes de 3 bits indiquant les niveaux de rouge, de vert et de bleu (cf. 1.1.2).

TRAP 14 fonction $8 : Floprd (a,b,c,d,e,f,g)

Lecture de secteur(s) sur un disque

paramètres d'appel : (long) adresse du tampon de stockage
(long) 0
(mot) numéro du lecteur (A=0, B=1)
(mot) numéro du secteur de départ (1 à 9)
(mot) numéro de la piste (0 à 79)
(mot) numéro de la face (0 ou 1)
(mot) nombre de secteurs à lire

paramètre de retour : 0 si lecture correcte, négatif sinon.

TRAP 14 fonction $9 : Flopwr (a,b,c,d,e,f,g)

Écriture de secteur(s) sur un disque

Les paramètres d'appel et de retour sont exactement semblables à ceux de la fonction précédente.

TRAP 14 fonction $A : Flopfmt (a,b,c,d,e,f,g,h,i)

Formatage d'une piste d'une face de disque

paramètres d'appel : (long) tampon de stockage (8 Ko minimum)
(long) 0
(mot) numéro du lecteur (A=0, B=1)
(mot) nombre de secteurs par piste
(mot) numéro de la piste à formater
(mot) numéro de la face à formater
(mot) facteur d'espacement entre secteurs
(long) $87654321 sur le 520 ST
(mot) mot de remplissage
paramètre de retour : 0 si formatage réussi, négatif sinon.

En cas de formatage défaillant, une liste des secteurs non formatés est retournée dans le tampon. Le quartet fort du mot de remplissage (dernier paramètre d'appel) ne doit pas être égal à $F. En dehors de cette restriction, le mot de remplissage peut être égal à n'importe quelle valeur, y compris 0.

TRAP 14 fonction $C : Midiws (a,b)

Envoi d'une chaîne sur le port MIDI

paramètres d'appel : (mot) nombre de caractères à envoyer – 1
(long) adresse de la chaîne de caractères
paramètre de retour : aucun

TRAP 14 fonction $D : Mfpint (a,b)

Place un vecteur d'interruption du 68901

paramètres d'appel : (mot) numéro du vecteur (0 à $F)
(long) adresse à placer dans le vecteur
paramètre de retour : aucun

L'adresse du vecteur est égale à : $100 + (4 * numéro du vecteur). Se reporter au chapitre 5 pour une analyse des vecteurs d'interruptions.

TRAP 14 fonction $E : Iorec (a)

Fournit l'adresse d'un descripteur de tampon d'entrée série

paramètres d'appel : (mot) 0 si série RS 232, 1 si clavier, 2 si MIDI
paramètre de retour : (long) adresse du descripteur de tampon d'entrée.

Le descripteur de tampon d'entrée à la structure :

0 (long) : adresse du tampon de stockage des entrées
4 (mot) : taille du tampon, en octets
6 (mot) : index du dernier octet entré
8 (mot) : index du dernier octet saisi
A (mot) : base du débordement
C (mot) : haut du débordement

Le haut du débordement est le nombre d'octets à partir duquel il y a envoi d'une demande d'arrêt de transmission (si le protocole XON/XOFF ou CTS/RTS est actif). Le bas du débordement est le nombre d'octets en dessous duquel la demande de reprise de transmission est envoyée (si un protocole est actif).

Pour l'entrée-série, un descripteur du tampon de *sortie série* est placé juste après le descripteur du tampon d'entrée (14 octets après donc).

TRAP 14 fonction $F : Rsconf (a,b,c,d,e,f)

Configuration du port série RS 232C

paramètres d'appel : (mot) code de la vitesse
(mot) code du protocole
(mot) fixe le registre de contrôle général
(mot) fixe le registre contrôle récepteur
(mot) fixe le registre contrôle émetteur
(mot) fixe registre caractère synchronisation.
paramètre de retour : (long) octet fort des quatre registres précédents (anciennes valeurs).

La vitesse est codée :

0 : 19200	4 : 2400	8 : 600	C : 134
1 : 9600	5 : 1920	9 : 300	D : 110
2 : 4800	6 : 1745	A : 200	E : 75
3 : 3840	7 : 1200	B : 150	F : 50

Le protocole est codé :

0 : pas de protocole (par défaut)	2 : RTS/CTS
1 : XON/XOFF (∧S et ∧Q)	3 : XON/XOFF et RTS/CTS

Le registre de contrôle général a la structure suivante :

bit 7 : doit être à 1 (division par 16 de l'horloge)
bit 5 et 6 : nombre de bits utiles (00 = 8 bits, 01 = 7 bits, 10 = 6 bits, 11 = 5 bits)
bits 3 et 4 : format de contrôle (00 = pas de START ni de STOP, 01 = 1 bit de START et 1 bit de STOP, 10 = 1 bit de START et 1,5 bit de STOP, 11 = 1 bit de START et 2 bits de STOP). Le format 00 est déconseillé.
bit 2 : validation de la parité (1 = parité valide, 0 sinon)
bit 1 : type de parité (1 = paire, 0 = impaire)
bit 0 : 0

Le registre de contrôle du récepteur a la structure :
bits 1 à 7 : en lecture (nuls en écriture)
bit 0 : valide le récepteur à 1, le désactive à 0

Le registre de contrôle du transmetteur a la structure :
bits 1 à 7 : en lecture (nuls en écriture)
bit 0 : valide l'émetteur à 1, le désactive à 0

Le registre de caractère de synchronisation : 0 sur le 520 ST

TRAP 14 fonction $10 : Keytbl (a,b,c)

Fixe les adresses des tables de décodage des touches du clavier

paramètres d'appel : (long) adresse de la table des minuscules
(long) adresse de la table des majuscules
(long) adresse de la table de décodage lorsque Caps-lock est enfoncée
paramètre de retour : (long) adresse ancienne table des minuscules

Les tables doivent comporter 128 octets, indiquant comment convertir un code clavier en code ASCII (voir table de base, Trap 1, fonction 1).

TRAP 14 fonction $11 : Random ( )

Fournit un nombre pseudo aléatoire

paramètres d'appel : aucun
paramètre de retour : (long) nombre aléatoire sur 24 bits

TRAP 14 fonction $12 : Protobt (a,b,c,d)

Fabrique une image du "boot" secteur d'un disque

paramètres d'appel : (long) adresse du tampon de stockage
(long) numéro de série (inchangé si −1, aléatoire si >$01000000)

(mot) type du disque (2 si 80 pistes, simple face, 3 si 80 pistes, double face)
(mot) rend le secteur exécutable si à 1, non exécutable si à 0, inchangé à −1.
paramètre de retour : (mot) somme de contrôle du secteur (checksum).

Permet de fabriquer une image du secteur 0, piste 1, pour écriture sur un disque ou modification du secteur 0 existant. Le tampon de stockage doit comporter 512 octets minimum.

TRAP 14 fonction $13 : Flopver (a,b,c,d,e,f,g,)

Vérification de secteur(s) d'un disque

paramètres d'appel : (long) adresse du tampon de stockage
(long) 0
(mot) numéro du disque (A=0, B=1)
(mot) numéro du secteur de départ à vérifier
(mot) numéro de la piste du disque (1 à 9)
(mot) numéro de la face du disque (0 ou 1)
(mot) nombre de secteurs à vérifier.
paramètre de retour : (long) 0 si vérification correcte, négatif sinon.

Le tampon de stockage doit avoir une adresse alignée sur un nombre entier de Ko. Au retour, il contient les numéros des secteurs éventuellement défaillants.

TRAP 14 fonction $14 : Prtblk ( ) SCRDMP

Copie d'écran sur imprimante

paramètres d'appel : aucun
paramètre de retour : (long) 0 si copie effectuée, négatif sinon.

La copie d'écran est effectuée en fonction des paramètres existants de configuration de l'imprimante.

TRAP 14 fonction $15 : Cursconf (a,b)

Fixe ou lit les attributs du curseur

paramètres d'appel : (mot) période de clignotement du curseur
(mot) attributs du curseur
paramètre de retour : (mot) selon paramètres d'appel

Le positionnement des attributs s'opère en fonction du 2e paramètre :
0 : désactivation du curseur
1 : activation du curseur

2 : rend le curseur clignotant (si actif)
3 : rend le curseur fixe (si actif)
4 : écriture de la période de clignotement du curseur
5 : lecture de la période de clignotement du curseur

Un paramètre de positionnement d'attributs supérieur à 5 n'est pas pris en compte. La "période de clignotement" correspond au nombre de cycles horloge entre chaque état du curseur (visible/non visible). Elle est fixée à 30 à l'initialisation du système soit .43 secondes environ de visibilité et donc une alternance toutes les 860 millisecondes.

TRAP 14 fonction $16 : Settime (a)

Fixe la date et l'heure.

paramètres d'appel : (long) date et heure
paramètre de retour : (long) date et heure placées

La date et l'heure sont formatées comme décrit en Trap 1, fonction $2A et $2C (Tgetdate et Tgettime).

TRAP 14 fonction $17 : Gettime ( )

Lit la date et l'heure

paramètres d'appel : aucun
paramètre de retour : (long) date et heure (cf. ci-dessus).

TRAP 14 fonction $18 : Bioskeys ( )

Réinitialise les tables de conversion de touches clavier système

Aucun paramètre d'appel ni de retour. Se reporter en TRAP 14, fonction $10 (Keytbl) pour le détail des tables de conversion.

TRAP 14 fonction $19 : Ikbdws (a,b)

Envoi de commandes au 6301 (microprocesseur clavier)

paramètres d'appel : (mot) nombre d'octets à envoyer
(long) adresse de la chaîne à envoyer
paramètre de retour : aucun

Se reporter au chapitre 5 pour le détail des commandes clavier.

TRAP 14 fonction $1A : Jdisint (a)

Désactive une interruption du 68901

paramètres d'appel : (mot) numéro de l'interruption (0 à $F)
paramètre de retour : (long) numéro de l'interruption.

Se reporter au chapitre 5 pour le détail des interruptions.

TRAP 14 fonction $1B : Jenabint (a)

Active une interruption du 68901

paramètres d'appel : (mot) numéro de l'interruption (0 à $F)
paramètre de retour : (long) numéro de l'interruption.

TRAP 14 fonction $1C : Giaccess (a,b)

Lit ou écrit un registre du synthétiseur de son

paramètres d'appel : (mot) donnée à écrire sur octet faible
(mot) numéro du registre (bit 7 à 0 pour la lecture, à 1 pour l'écriture)
paramètre de retour : (octet) donnée lue ou écrite dans le registre.

cf. chapitre 1, le synthétiseur de son (1.1.3) et chapitre 5 pour des explications détaillées sur les registres. *Attention* aux écritures sur le port A (registre $E) qui peuvent être funestes...

TRAP 14 fonction $1D : Offgibit (a)

Positionnement d'un bit à 0 sur le port A du son.

paramètres d'appel : (mot) numéro du bit à annuler
paramètre de retour : aucun

cf. 1.1.3. pour la fonction de chacun des bits du port A.

TRAP 14 fonction $1E : Ongibit (a)

Positionnement d'un bit 1 sur le port A du circuit son

paramètres d'appel : (mot) numéro du bit (voir ci-dessus)
paramètre de retour : aucun

TRAP 14 fonction $1F : Xbtimer (a,b,c,d)

Initialisation d'un timer du 68901

paramètres d'appel : (mot) numéro du timer (0 à 3)
(mot) registre de contrôle du timer
(mot) registre de données du timer
(long) adresse de la routine d'interruption
paramètre de retour : (long) numéro de l'interruption timer

Quatre "timers" sont implementées sur le 68901 :
A (0) : disponible
B (1) : utilisé par le système pour rafraîchir l'écran
C (2) : base de temps système réglée à 200 Hz
D (3) : vitesse série (voir 1.1.3).

Pour le timer A, seul disponible, le registre de contrôle a les fonctions suivantes :
bits 0 à 2 : division de l'horloge si bit 3 à 0 (voir 1.1.3)
compteur d'événements si bits 0 à 4 = $8
générateur de signaux si bits 0 à 4 > $8.
bit 4 : timer stoppé si à 0, actif si à 1
bits 5 à 7 : inutilisés

Le registre de données fournit la base de temporisation, il est decrémenté à chaque cycle de la base de temps. Arrivé à 1, une interruption est générée et le programme est dérouté vers la routine d'interruption pointée par le dernier paramètre d'appel. Puis le registre de données est rechargé avec sa valeur, etc.

TRAP 14 fonction $20 : Dosound (a)

Envoi de commande(s) au synthétiseur de son

paramètres d'appel : (long) adresse des commandes à envoyer
paramètre de retour : (long) ancien pointeur des commandes.

Voir chapitre 5 pour le détail des commandes

TRAP 14 fonction $21 : Setprt (a)

Fixe ou lit la configuration imprimante

paramètres d'appel : (mot) configuration (lecture si −1)
paramètre de retour : (mot) configuration de l'imprimante.

La configuration de l'imprimante est fixée ainsi :
bit 0 : 0 si matricielle, 1 si marguerite
bit 1 : 0 si monochrome, 1 si couleur
bit 2 : 0 si 1280 points/ligne, 1 si 960 points/ligne
bit 3 : qualité épreuve si 0, définitive si 1
bit 4 : 0 si sortie parallèle, 1 si sortie série
bit 5 : 0 si listing, 1 si feuille
bits 6 à 15 : inutilisés (le bit 15 doit être nul).

TRAP 14 fonction $22 : Kbdvbase ( )

Demande l'adresse de base du descripteur de vecteurs clavier

paramètres d'appel : aucun
paramètre de retour : (long) adresse de base du descripteur

L'adresse retournée pointe sur la structure suivante :

0 (long) adresse routine d'entrée MIDI
4 (long) adresse routine traitement erreur clavier (rts !)
8 (long) adresse routine traitement erreur MIDI (rts !)
C (long) adresse routine de test clavier (rts !)
10 (long) adresse routine d'interruption souris
14 (long) adresse routine d'interruption horloge
18 (long) adresse routine d'interruption manette de jeu

TRAP 14 fonction $23 : Kbrate (a,b)

Fixe ou lit la fréquence d'autorépétition des touches du clavier

paramètres d'appel : (mot) délai avant autorépétition
(mot) délai entre auto-répétitions
paramètre de retour : (mot) octet fort = délai avant autorépétition,
octet faible = délai entre autorépétitions.

Les délais sont exprimés en multiples de la base de temps, soit 20 ms. Si l'un des paramètres est égal à −1, il est lu et non écrit.

TRAP 14 fonction $24 PRTBLK

Copie d'écran sur imprimante paramétrable

paramètres d'appel : (long) adresse du bloc de paramètres
paramètre de retour : 0 si copie, −1 si erreur.

Le bloc de paramètres doit comporter au moins 30 octets. Il doit avoir la structure suivante :

0 (long) adresse du début de la mémoire d'écran à copier
4 (mot) 0
6 (mot) nombre de points horizontaux à copier
8 (mot) nombre de points verticaux à copier
A (mot) 0
C (mot) nombre de points horizontaux de l'écran
− nombre de points horizontaux à copier
E (mot) résolution (2 = haute, 1 = moyenne, 0 = basse)
10 (mot) qualité (0 = épreuve, 1 = définitif)
12 (long) adresse de la palette de couleurs
16 (mot) 0 si matricielle noir et blanc 1280 points/ligne
1 si matricielle couleur 1280 points/ligne

2 si marguerite noir et blanc Atari
3 si matricielle noir et blanc 960 points/ligne
−1 sinon (pas de copie d'écran...)

18 (mot) 0 si sortie parallèle, 1 si sortie série

1A (long) adresse table de conversion des couleurs (table de 16 octets, dont les 4 bits faibles traduisent chaque couleur)

Attention, cette fonction (ainsi que les fonctions $26 et $27) ne sont pas documentées et leur utilisation réclame quelques précautions. Ainsi cette copie d'écran ne fonctionnera que si l'on a placé auparavant 1 dans la variable système $4EE (drapeau de copie).

TRAP 14 fonction $25 WVBL

Attente de la prochaine interruption "verticale"

Aucun paramètre. La routine appelée attend simplement l'interruption fréquence de trame. A utiliser pour la synchronisation graphique.

TRAP 14 fonction $26 SUPEXEC

Exécution d'une routine en mode Trap

paramètres d'appel : (long) adresse de la routine à exécuter
paramètre de retour : selon votre bon plaisir

Permet de faire exécuter une routine que vous avez écrite comme un Trap, c'est-à-dire en mode superviseur et sans trace. Cette routine doit se terminer par un rts (le retour d'interruption est assuré par le système). Les registres D3 à D7 et A3 à A7 sont restitués inchangés.

TRAP 14 fonction $27 PUNTAES

Suppression de l'AES du système

Aucun paramètre. Fonction sans doute inutilisable une fois que le GEM sera en mémoire morte... A manipuler avec une extrême précaution, certaines modifications étant indispensables (par exemple une désactivation de la ligne F, cf. chapitre 3).

ANNEXE 5
INSTRUCTIONS DU 68000

L'ensemble des instructions du 68000 est ici fourni dans l'ordre alphabétique. Les abréviations suivantes sont utilisées :

<Donnée>	désigne une donnée immédiate
Dn	désigne un registre de donnée
An	désigne un registre d'adresse
USP	désigne le pointeur de pile utilisateur
SR	désigne le registre d'état (16 bits)
CCR	désigne le registre de condition (octet faible de SR)
Ds	désigne un registre de donnée source
Dd	désigne un registre de donnée destination
As	désigne un registre d'adresse source
Ad	désigne un registre d'adresse destination
EA	désigne une Adresse Effective d'une position mémoire
B, W, L	correspondent aux tailles Octet, Mot et Long mot

Dans la colonne de droite, modes d'adressage, les adressages autorisés sont marqués d'une croix. On a :

Dn	adressage registre de donnée
An	adressage registre d'adresse
I	adressage indirect
I+	adressage indirect post-incrémenté
−I	adressage indirect pré-décrémenté
dI	adressage indirect avec déplacement
Ab	adressage absolu (court ou long)
R	adressage relatif à PC (avec ou sans déplacement)
Im	adressage immédiat

X, N, Z, V, C, correspondent aux indicateurs du registre de conditions (pour ces indicateurs, Ø ou 1 correspond à un positionnement obligatoire, la lettre P correspond à un positionnement en fonction du déroulement de l'instruction et la lettre I à un positionnement indéfini).

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
ABCD	Ds	Dd	$(Ds)_{10} + (Dd)_{10} + X \rightarrow Dd_{10}$	8 bits	X									P	I	P	I	P
	−(As)	−(Ad)	$(EAs)_{10} + (EAd)_{10} + X \rightarrow EAd_{10}$	8 bits					X									
ADD	source	Dn	(source) + (Dn) → Dn	B,W,L	X	X	X	X	X	X	X	X	X	P	P	P	P	P
	Dn	EA	(Dn) + (EA) → EA				X	X	X	X	X							
ADDA	source	An	(source) + (An) → An	W,L	X	X	X	X	X	X	X	X	X	aucun				
ADDI	<donnée>	dest.	<donnée> + (dest.) → dest.	B,W,L	X		X	X	X	X	X	X	X	P	P	P	P	P
ADDQ	<donnée>	dest.	(1 à 8) + (dest.) → dest.	B,W,L	X	X	X	X	X	X	X	X	X	P	P	P	P	P
ADDX	Ds	Dd	(Ds) + (Dd) + X → Dd	B,W,L	X									P	P	P	P	P
	−(As)	−(Ad)	(EAs) + (EAd) + X → EAd						X									
AND	source	Dn	source ET (Dn) → Dn	B,W,L	X		X	X	X	X	X	X	X		P	P	0	0
	Dn	EA	(Dn) ET (EA) → EA				X	X	X	X	X							
ANDI	<donnée>	dest.	<donnée> ET (dest.) → dest.	B,W,L	X		X	X	X	X	X	X	X		P	P	0	0
ANDI to CCR	<donnée 8 bits>	CCR	<donnée> ET CCR → CCR	8 bits										pos. inst.				
ANDI to SR	<donnée 16 bits>	SR	<donnée> ET SR → SR	16 bits										pos. instr.				
ASL	Ds	Dd	C ← opérande ← 0	B,W,L	X									P	P	P	P	P
	<donnée>	Dn	X ←	B,W,L	X													
	EA			W			X	X	X	X	X							
ASR	Ds	Dd	→ opérande → C	B,W,L	X									P	P	P	P	P
	<donnée>	Dn	→ X	B,W,L	X													
	EA			W			X	X	X	X	X							

Branchement	Condition de branchement	Nombres	Codes de conditions
BCC	retenue à zéro	non signés	C = Ø
BCS	retenue à un	non signés	C = 1
BEQ	égal	signés ou non signés	Z = 1
BNE	différent	signés ou non signés	Z = Ø
BGE	supérieur ou égal	signés	N = Ø et V = Ø, ou N = 1 et V = 1
BGT	supérieur	signés	Z = Ø, et N = Ø et V = Ø Z = Ø et N = 1 et V = 1
BHI	supérieur	non signés	C = Ø et Z = Ø
BLE	inférieur ou égal	signés	Z = 1, et N = 1 et V = Ø Z = 1, et N = Ø et V = 1
BLS	inférieur ou égal	non signés	C = 1 ou Z = 1
BLT	inférieur	signés	N = 1 et V = Ø, ou N = Ø et V = 1
BMI	négatif		N= 1
BPL	positif		N = Ø
BVC	débordement à Ø		V = Ø
BVS	débordement à 1		V = 1
BRA	toujours		

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
BCHG	Dn	dest.	teste le bit, inverse sa valeur et place le complément dans Z	8 bits si dest. mémoire			X	X	X	X	X					P		
	<donnée>	dest.			X		X	X	X	X	X					P		
BCLR	Dn	dest.	place le complément du bit dans Z et met le bit à 0	L si dest = Dn			X	X	X	X	X					P		
	<donnée>	dest.			X		X	X	X	X	X					P		
BSET	Dn	dest.	place le complément du bit dans Z et met ce bit à 1				X	X	X	X	X					P		
	<donnée>	dest.			X		X	X	X	X	X					P		
BSR	EA		appel de sous-programme									X						
BTST	Dn	dest.	place le complément du	voir			X	X	X	X	X					P		
	<donnée>	dest.	bit dans Z	ci-dessus	X		X	X	X	X	X					P		
CHK	source	Dn	exception si (Dn) 0 ou (Dn) (source)	16 bits	X		X	X	X	X	X	X	X		P	I	I	I
CLR	dest.		0 → dest.	B,W,L	X		X	X	X	X	X				0	1	0	0
CMP	source	Dn	teste (Dn) − (source)	B,W,L	X	X	X	X	X	X	X	X	X		P	P	P	P
CMPA	source	An	teste (An) − (source)	W,L	X	X	X	X	X	X	X	X	X		P	P	P	P
CMPI	source	dest.	teste (dest.) − (donnée)	B,W,L	X		X	X	X	X	X	X	X		P	P	P	P

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
CMPM	(As) +	(Ad) +	teste (EAd) − (EAs)	B,W,L				×							P	P	P	P
DBcc	Branchement si condition réalisée ou si le compteur Dn (16 bits) diffèrent de −1											×						
DIVS	source	Dn	(Dn)/(source) → (Dn)	32/16 signés	×		×	×	×	×	×	×	×		P	P	P	0
DIVU	source	Dn	(Dn)/(source) → (Dn)	32/16	×		×	×	×	×	×	×	×		P	P	P	0
EOR	Dn	dest.	(Dn) EOR (dest.) → (dest.)	B,W,L	×		×	×	×	×	×	×	×		P	P	0	0
EORI	<donnée>	dest.	donnée EOR (dest) → (dest)	B,W,L	×		×	×	×	×	×	×	×		P	P	0	0
EORI to CCR	<donnée>	CCR	donnée EOR (CCR) → (CCR)	8 bits											pos. inst.			
EORI to SR	<donnée>	SR	donnée EOR (SR) → (SR)	16 bits											pos. inst.			
EXG	Rs	Rd	(Rs) ⟷ (Rd)	32 bits	×	×									P	P	0	0
EXT	Dn		extension de signe à 16 ou 32 bits	W,L	×										P	P	0	0
JMP	EA		PC = EA				×			×	×	×						
JSR	EA		appel de sous-programme à l'adresse EA				×			×	×	×						
LEA	EA	An	EA → An	32 bits			×			×	×	×						
LINK	AN → − (SP) SP → An SP + d → SP		le contenu du registre d'adresse An est empilé. Puis ce registre d'adresse est chargé avec la valeur du pointeur de pile. Enfin, le déplacement (étendu sur 32 bits) est ajouté au pointeur de pile.															

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
LSL	Ds	Dd	C opérande Ø	B,W,L	X									P	P	P	Ø	P
	<donnée>	Dn		B,W,L	X													
	EA		X	W			X	X	X	X	X							
LSR	Ds	Dd	Ø opérande C	B,W,L	X									P	P	P	Ø	P
	<donnée>	Dn		B,W,L	X													
	EA		X	W			X	X	X	X	X							
MOVE	source	destin.	(source) → (dest.)	source →	X	X	X	X	X	X	X	X	X		P	P	Ø	Ø
				B,W,L	X	X	X	X	X	X	X	X	X					
				dest. →	X		X	X	X	X	X	X	X					
MOVE → CCR	source	CCR	(source) → CCR	8 bits	X		X	X	X	X	X	X	X	pos. inst.				
MOVE de SR	SR	dest.	(SR) → dest.	16 bits	X		X	X	X	X	X	X	X					
MOVE → SR	source	SR	(source) → SR	16 bits	X		X	X	X	X	X	X	X	pos. inst.				
MOVE USP	USP	An	(USP) → An	L		X												
	An	USP	(An) → USP			X												
MOVEA	source	An	(source) → An	W,L	X	X	X	X	X	X	X	X	X					
MOVEM	liste	dest.	liste reg. → dest.	W,L			X		X	X	X							
	source	liste	(source) → liste reg.	W,L			X	X		X	X							
MOVEP	Dn	d (An)	(Dn) → EA = (An) + d	W,L						X								
	d (An)	Dn	(EA) → Dn	W,L						X								
MOVEQ	donnée	Dn	donnée → Dn	32 bits	X										P	P	Ø	Ø

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
MULS	source	Dn	(source) × (Dn) → Dn signés	16 × 16	X		X	X	X	X	X	X	X		P	P	0	0
MULU	source	Dn	(source) × (Dn) → Dn	16 × 16	X		X	X	X	X	X	X	X		P	P	0	0
NBCD	dest.		complément à 9 si X = 1 à 10 si X = 0	8 bits	X		X	X	X	X	X	X	X	P	I	P	I	P
NEG	dest.		complémentation à 2	B,W,L	X		X	X	X	X	X			P	P	P	P	P
NEGX	dest.		complément à 2 avec X	B,W,L	X		X	X	X	X	X			P	P	P	P	P
NOP			aucune opération															
NOT	dest.		complémentation à 1	B,W,L	X		X	X	X	X	X				P	P	0	0
OR	Dn	dest.	(Dn) OU (dest) → (dest)	B,W,L			X	X	X	X	X				P	P	0	0
	source	Dn	(source) OU (Dn) → (Dn)	B,W,L	X		X	X	X	X	X	X	X		P	P	0	0
ORI	donnée	dest.	donnée OU (dest) → (dest)	B,W,L	X		X	X	X	X	X				P	P	0	0
ORI → CCR	donnée	CCR	donnée OU CCR → CCR	8 bits										pos. inst.				
ORI → SR	donnée	SR	donnée OU SR → SR	16 bits										pos. inst.				
PEA	EA		empile une adresse effective	32 bits			X			X	X	X						
RESET			validation de la ligne Reset (en mode Superviseur). En mode utilisateur, génère une exception.															

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	−I	dI	Ab	R	Im	X	N	Z	V	C
ROL	Ds	Dd	C opérande	B,W,L	X										P	P	Ø	P
	<donnée>	Dn		B,W,L	X										P	P	Ø	P
	EA		décalage d'un bit	W			X	X	X	X	X				P	P	Ø	P
ROR	Ds	Dd	opérande C	B,W,L	X										P	P	Ø	P
	<donnée>	Dn		B,W,L	X										P	P	Ø	P
	EA		décalage d'un bit	W			X	X	X	X	X							
ROXL	Ds	Dd	C opérande	B,W,L	X									P	P	P	Ø	P
	<donnée>	Dn	X	B,W,L	X									P	P	P	Ø	P
	EA		décalage d'un bit	W			X	X	X	X	X			P	P	P	Ø	P
ROXR	Ds	Dd	opérande C	B,W,L	X									P	P	P	Ø	P
	<donnée>	Dn	X	B,W,L	X									P	P	P	Ø	P
	EA		décalage d'un bit	W			X	X	X	X	X			P	P	P	Ø	P
RTE	(SP.W) + → SR (SP.L) + → PC		retour d'exception															
RTE	(SP.B) + → CCR (SP.L) + → PC		retour et restitution du code condition															
RTS	(SP.L) + → PC		retour de sous-progr.															
SBCD	Ds	Dd	$(Dd)_{10} - (Ds)_{10} - X \rightarrow Dd$	8 bits	X									P	I	P	I	P
	− (As)	− (Ad)	$(EAd)_{10} - (EAs)_{10} - X \rightarrow EAd$	8 bits					X									
Scc	voir Bcc		si condition vraie, $FF → dest. Sinon Ø → dest	8 bits	X		X	X	X	X	X							

Instruction	Source	Destin.	Opération effectuée	Taille	Modes d'adressage									Indicat.				
					Dn	An	I	I+	-I	dI	Ab	R	Im	X	N	Z	V	C
STOP	donnée	SR	arrêt du 68008	16 bits														
SUB	source	Dn	(Dn) – (source) → Dn	B,W,L	X	X	X	X	X	X	X	X	X	P	P	P	P	P
	Dn	EA	(EA) – (Dn) → EA	B,W,L			X	X	X	X	X							
SUBA	source	An	An) – (source) → An	W,L	X	X	X	X	X	X	X	X	X					
SUBI	donnée	dest.	(dest) – (donnée) → dest	B,W,L	X		X	X	X	X	X			P	P	P	P	P
SUBQ	donnée (1 à 8)	dest.	(dest) – (donnée) → dest.	B,W,L	X	X	X	X	X	X	X			P	P	P	P	P
SUBX	Ds	Dd	(Dd) – (Ds) – X → Dd	B,W,L	X									P	P	P	P	P
	– (As)	– (Ad)	(EAd) – (EAs) – X → EAd						X									
SWAP	Dn		échange le mot fort et le mot faible de Dn	32 bits	X										P	P	0	0
TAS	EA		test d'un octet et mise à 1 du bit 7	8 bits	X		X	X	X	X	X				P	P	0	0
TRAP	numéro		déroutement vers routine															
TRAPV			déroutement vers exception débordement si V = 1															
TST	dest.		teste la destination	B,W,L	X		X	X	X	X	X				P	P	0	0
UNLK	An → SP (SP)+ → An		restitution de contexte (voir LINK)															

BIBLIOGRAPHIE

Les commentaires accompagnant les titres de cette bibliographie constituent des jugements subjectifs de l'auteur, qui a toutefois pris le soin de lire et souvent de relire tous ces livres.

1. Ouvrages sur le 520 ST.

Le nouvel ATARI ST, auteur inconnu. Ed. MICRO APPLICATION. Paris, 1985. Ouvrage dont l'unique intérêt est d'avoir été le premier. Beaucoup d'imprécisions mais des notes intéressantes sur les entrées-sorties.

Le livre du langage machine de l'ATARI ST, auteur inconnu. Ed. MICRO APPLICATION. Paris, 1985. Niveau et présentation satisfaisantes mais aucun détail sur le 520 ST et la programmation des fonctions système. Programmes pour hyper débutant.

2. Ouvrages sur le langage C.

Le langage C, B.W. KERNIGHAN & D.M. RITCHIE. Ed. MASSON. Paris, 1984. La "bible" du programmeur en C. Écrit par les pères du langage C, indispensable.

Langage C, Philippe DAX. Ed. EYROLLES. Paris, 1984. Bon complément du précédent, clair et illustré d'exemples.

Langage C, J.L. FOURTANIER & V. PRINCE. Ed. DITEST. Paris, 1984. Simple, parfois simpliste, passe à côté des difficultés. Trop succint.

Programmer en C, C. NOWAKOWSKI. Ed. du PSI. Lagny, 1984. Nombreux exemples de programmes, délicats à adapter sur le ST mais intéressants.

C par l'exemple, J.M. DRAPPIER & A. MAUFFREY. Ed. EYROLLES. Paris, 1986. Longs programmes malheureusement un peu confus et difficiles à adapter au ST.

3. Ouvrages sur l'assembleur 68000

MC 68000 16-bit MICROPROCESSOR, User's Manual. MOTOROLA, 1981. Comment s'en passer lorsqu'on comprend un tant soi peu l'anglais ?

Le microprocesseur 68000 et sa programmation, P. JAULENT. Ed. EYROLLES. Paris, 1984. Bonne présentation des interruptions, exemples de programmes.

Mise en œuvre du 68000, C. VIEILLEFOND. Ed. SYBEX. Paris, 1984. Manuel exhaustif des instructions. Beaucoup d'informations, peu de programmes.

68000 Microprocessor Handbook, G. KANE. Ed. OSBORNE. Berkeley, 1981. Tableaux des modes d'adressage et des temps d'exécution en cycles très complets.

Le désassemblage de la ROM du QL, L. de BURSAC. Ed. LOGISOFT. Toulouse, 1985. Un ouvrage unique pour les fanas de l'assembleur : 44 Ko de routines écrites en assembleur 68000 listées et commentées.

Programmation en assembleur du 68000 et des circuits d'interface, F. Brown. EDITESTS. Paris, 1985. Assez confus. Peu adapté à la programmation sur ST. Programmes simplistes et peu optimisés.

Le microprocesseur MC68000, contrôle et mise au point des composants associés, J.W. COFFRON. Ed. MASSON. Paris, 1985. Bien fait mais sans utilité pour programmer sur ST.

Circuits périphériques de la famille 68000, P. JAULENT. Ed. EYROLLES. Paris, 1985. Ouvrage de référence en ce qui concerne le 68901.

Programmation en Assembleur 68000. Léon. Ed. Eyrolles, 1986.

INDEX

A

A (ligne) 80-97
abort 139
abs 139
access 139
Acessoires 46-55, 113-115
addr_in 69, 110, 190
addr_out 69, 110, 190
AES 37, 189-226
appl_exit 193
appl_find 192
appl_init 190
appl_read 191
appl_tplay 193
appl_trecord 192
appl_write 191
Assembleur 61-100
atan 148
atof, atoi, atol 140

B

Basic 106-111
Bconin 251
Bconout 252
Bconstat 251
Bcostat 253
Bibliothèques C 35-36, 103
Bioskeys 261
BREAK 27, 29
brk 140

C

C (langage) 17-60
Calculatrice 46-55
calloc 140
CASE 30
Cauxin 236
Cauxis 238
Cauxos 239
Cauxout 236
Cconin 235
Cconis 238
Cconos 238
Cconout 235
Cconrs 237
Cconws 237
ceil 140
chmod. chown 140
Clavier 8, 121-130
clearerr 142
close 141
CMP 67
Cnecin 237
codes clavier 235
Compilation 31, 95, 101-106
Constantes C 19
CONTINUE 27-28
contrl 68, 107, 149-150
control 69, 190
coordonnées 60
cos 141
couleurs 6-7
CP/M 12

Cprnos 238
Cprnout 236
Crawcin 237
Crawio 236
creat 141
Cursconf 260

D

Dbcc 63
Dcreate 242
Ddelete 242
Déclarations 20-21
DEFAULT 30
Démarrage 42
Dfree 241
Dgetdrv 239
Dgetpath 246
Dialogue 45
Disque virtuel 74-79
DIV 66
DO-WHILE 29
Dosound 263
Drvmap 253
Dsetdrv 238
Dsetpath 242

E

Erreurs 55-60, 105
ESCAPE 68, 178
etoa 141
Evénement 43-44, 112
evnt_button 194
evnt_dclick 200
evnt_keybd 194
evnt_mesag 195
evnt_mouse 195
evnt_multi 198
evnt_timer 198
Exceptions (v. Traps)
EXG 67
Extension (bit) 62-63

F

F (ligne) 98-100
fabs 141
Fattrib 245
Fclose 244
fclose 142
Fcreate 243
Fdatime 249
Fdelele 244
fdopen 142
Fdup 245
feof 142
ferror 142
fflush 142
Fforce 246
Fgetdta 241
fgets 143
FILE (structure) 57, 59
fileno 142
floor 142
Flopfmt 257
Floprd 256
Flopver 260
Flopwr 257
fmod 142
fontes (v. polices)
Fopen 243
fopen 142
FOR 30
form_alert 208
form_center 209
form_dial 207
form_do 207
form_error 208
fprintf 145
fputc 145
fputs 145
Fread 244
fread 143
free 140
Frename 248
freopen 142
fscanf 146
Fseek 245
fseek 143
fsel_input 225
Fsetdta 239
Fsfirst 248
Fsnext 248
ftell 143
ftoa 141
Fwrite 244
fwrite 143

G

GB 109-110
GEM 36-46, 106-111
GEMSYS 107, 110-111
getc 143
getchar 143
getl 143
getpass 143
Getrez 255
gets 143
Getshift 254
Gettime 261
getw 143
Giaccess 262
global 69, 110, 190
GOTO 28
graf_dragbox 210
graf_growbox 211
graf_handle 212
graf_mkstate 213
graf_mouse 213
graf_movebox 210
graf_rubberbox 209
graf_shrinkbox 211
graf_slidebox 212
graf_watchbox 211
Graphisme 8

H

Horloge 126

I

Identificateurs 19
IF-ELSE 29
Ikbdws 261
index 144
Initmous 254
Instructions C 24-31
Instructions 68000 267-276
Interruptions 117-135, 228-229
intin 68, 107, 149
int_in 69, 110, 190
intout 68, 107, 149
int_out 69, 110, 190
Iorec 257
isatty 144

J

Jdisint 261
Jenabint 262

K

Kbdvbase 263
Kbrate 264
Keytbl 259

L

Langages 13-16, 101-111
Liens (édition) 101-106
LINK 65
log 144
Logbase 255
longjump 146
lseek 144

M

main 22
Malloc 246
malloc 140
manettes de jeu 124-125
matrices (remplissage) 86-87, 162
Mediach 253
menu_bar 200
menu_icheck 201
menu_ienable 201
menu_register 202
menu_text 202
menu_tnormal 202
Mfpint 257
Mfree 247
Midiws 257
mktemp 144
MOVEP 64-65
Mshrink 247
multitache 112-113

O

objc_add 203
objc_change 206

objc_delete 203
objc_draw 206
objc_edit 203
objc_find 204
objc_offset 205
objc_order 205
Objets 38-42
Offgibit 262
Ongibit 262
open 144
Operateurs C 21-24

P

parallèle (sortie) 10-11
perror 145
Pexec 247
Physbase 255
pile 64, 70
plans 7
POKE 108-109
pointeurs 33-34
polices de caractères 83-84
pow 145
printf 18, 145
programmation 13-14
    méthodes : 61-62
    (voir erreurs)
Protobt 259
Prtblk 260
Pterm 247
PtermO 235
Ptermres 241
Ptsin 68, 107, 149
Ptsout 68, 107, 149
putc 145
putchar 145
putl 145
puts 145
putw 145

Q

qsort 145

R

rand 146
Random 259
read 146
realloc 140
Résolution 5-6
Ressources 38-42
Retraçage 45-46
RETURN 26
rewind 143
rindex 144
Rsconf 258
rsrc_free 221
rsrc_gaddr 220
rsrc_load 220
rsrc_obfix 221
rsrc_saddr 221
Rwabs 252

S

sbrk 140
scanf 146
Scc 64-65
scrp_read 226
scrp_write 225
Série (E/S) 10-11, 121-122
Setcolor 256
Setexc 252
Setjmp 146
Setpalette 256
Setprt 263
Setscreen 256
Settime 261
shel_envrn 223
shel_find 223
shel_get 223
shel_put 224
shel_read 222
shel_write 222
signal 147
SIZEOF 24
son 8-10, 130-135, 231
SOUND 131
Souris 92, 122-124
Spirites 92-93
sqrt 147
srand 146
sscanf 146
strcat 147
strcmp 147
strcpy 147

strlen 147
Structuration 69-72
Structures C 34-35
Super 239
Superviseur (mode) 4
Sversion 241
swab 147
SWITCH 30
Synthétiseur (v. son)

T

Tableaux C 32-33
tan 148
Tgetdate 240
Tgettime 240
TOS 12-13
Traps 13, 233-265
TRAP 1 235-249
TRAP 2 249-250
TRAP 13 250-254
TRAP 14 136-137, 254-265
Tsetdate 240
Tsettime 240
ttyname 148
types C 20

U

ungetc 148
unlink 148
UNLK 65
Utilisateur (mode) 4

V

v_arc 159
v_bar 158
v_circle 159
v_clrwk 154
v_clsvwk 154
v_clswk 151
v_contourfill 94, 164
v_curdown 176
v_curhome 176
v_curleft 176
v_curright 176
v_curtext 177
v_curup 176
VDI 36-37, 149-189
VDISYS 107-108
v_dspcur 180
v_eeol 176
v_eeos 176
v_ellarc 160
v_ellipse 160
v_ellpie 160
v_enter_cur 175
vex_butv 189
vex_curv 189
v_exit_cur 178
vex_motv 189
vex_timv 189
v_fillarea 164
v_get_pixel 168
v_gtext 172
v_hardcopy 188
v_hide_c 179
Vitesse programmes 14-15
Vitesse série 10-11
v_justified 172
v_opnvwk 151
v_opnwk 150
v_pieslice 159
v_pline 157
v_pmarker 166
vq_chcells 178
vq_color 168
vq_curadress 177
vq_extnd 153
vqf_attributes 164
vq_key_s 183
vql_attributes 158
vqm_attributes 166
vq_mouse 182
vqt_attributes 174
vqt_extent 173
vqt_fontinfo 186
vqt_name 186
vqt_width 173
vr_box 158
vrf_box 158
vrm_cur 180
vro_cpyfm 89, 183
vrq_locator 181
vrq_string 182
vr_recfl 158
vrt_cpyfm 89, 184
vr_trnfm 185

v_rvoff 176
v_rvon 176
vsc_form 179
vs_clip 154
vs_color 167
vs_curadress 177
vsf_color 167
vsf_interior 86, 161
vsf_perimeter 163
vsf_style 86, 161
vsf_udpat 86, 163
v_show_c 179
vsin_mode 181
vsl_color 167
vsl_ends 157
vsl_type 155
vsl_udsty 156
vsl_width 156
vsm_color 167
vsm_height 166
vsm_type 165
vst_alignment 169
vst_color 167
vst_effects 171
vst_font 186
vst_height 91, 170
vst_load_fonts 68, 185
vst_point 91, 188
vst_rotation 171
vst_unload_fonts 187
vswr_mode 85, 155

W

WAVE 131-132
WHILE 29
wind_calc 218
wind_close 219
wind_create 214
wind_delete 219
wind_find 217
wind_get 215
wind_open 215
wind_set 216
wind_update 218
write 148

X

Xbtimer 262

2149 8-9
6301 11, 121-130
68000 1-5
68901 119-121, 231-232

Photocomposition : COMPO 2000 - SAINT-LO
Imprimerie de la Manutention à Mayenne
Dépôt légal : avril 1987
N° d' Editeur : 4635

Cet ouvrage vous fait pénétrer au coeur du ST, à travers une étude précise du TOS et du GEM, illustrée de nombreux exemples. Parmi ceux-ci, une calculatrice et un DISQUE VIRTUEL, listés et commentés, vous permettront de disposer de deux applications utiles.

Les méthodes et les erreurs de programmation, tant en langage C qu'en assembleur 68000, font l'objet de développements importants. Au programmeur confirmé, une analyse des lignes «A» et «F» ainsi qu'une analyse du circuit clavier ouvriront des perspectives nouvelles.

En annexes, toutes les primitives système et les fonctions GEM sont étudiées une par une, un index détaillé permettant une utilisation optimale des informations ainsi disponibles.

EYROLLES

8397